2017/07/21

How to describe a parameter file

E1 Introduction

This document explains how to describe LIBSDNN parameter files, as shown in Figure E-1. For a description of the SDNN and its terms, please refer to **Selective Desensitization Neural Networks**.

E2 Basics of parameter description

- The parameter file must be written in a code using ASCII characters such as latin1 or JIS; note that UTF-8 code cannot be used for file description. The parameter file must have a .txt extension.
- All parameters must be written on a single line in the form “parameter’s name = parameter’s content”.

```
<ISSUE> %This is an example of a parameter file
{
        type = function_approximation
        <FA>
        {
                output_range = [-0.2, 1.2]
                required_step_size = 0.005
        }
        input_number = 2
}
<SDNN>
{
        <PC>
        {
                n = 2000
                input_type_and_creation_method=[NUMERICAL(RANDOM_INVERSE(1001,5)):2]
                random_seed = hardware_entropy
        }
        <SD>
        {
                combination_setting = mutual
        }
        <NN>
        {
                random_seed = hardware_entropy
                initial_value_range = [-5, 5]
        }
}
<APP>
{
        print_progression = Y
        multi_thread = Y
        thread_number = 6
}
```

Fig. E-1: Example of a parameter file
for a two-variable function approximation.

- The content of a parameter can be a number, vector, character string, or character string array.
- Number and character strings can be described using the numbers or strings themselves.

- Vectors and character string arrays can be described in the form [element1, element2, ...]. If a given value is repeated, the individual element descriptions can be skipped by using the form "content: repetition number." For example, [1, 1, 2, 3, 3, 3] can be described using [1: 2, 2, 3: 3].
- A hierarchical structure is implemented using tags enclosed by < > and scopes enclosed by {}.
- Use a '%" to create a comment that can be extended to the end of a line
- Parameters that are not set by the user are automatically assigned their default values.
- If an unsuitable parameter is set, the application will request that you re-enter the parameter through the console.

E3 Parameter Details

In this section, we describe the tags and parameters used in LIBSDNN. All tags and parameters are formatted as tag1\tag2\...\parameter.

E3-1 ISSUE Parameters

E3-1-1 ISSUE\type

The ISSUE parameter type is either pattern recognition or function approximation.

- ◆ The following are acceptable parameter formats: pattern_recognition (for handling pattern recognition issues);
- ◆ function_approximation (for handling function approximation issues);

The default type is:

function_approximation

The parameter issue type is set as follows: for pattern recognition, use the parameter tag "ISSUE\PR"; for function approximation, use the tag "ISSUE\FA".

E3-1-1-1 ISSUE\PR\class_number

This sets the number of classes for a pattern recognition issue.

Acceptable format:

An integer greater than or equal to 2.

Default:

2

E3-1-1-2 ISSUE\PR\multi_class_recognition

This sets the multi-class recognition method as either a one-versus-one or a one-versus-rest classifier. Note that the one-versus-one classifier requires more time/memory but often performs more accurately than the one-versus-rest classifier.

Acceptable formats:

- ◆ 1v1 (one-versus-one)
- ◆ 1vR (one-versus-rest)

Default:

1v1

E3-1-1-3 ISSUE\FA\output_range

The output range of the approximated function.

Acceptable format:

A two-element-vector in the form [minimum_value, maximum_value].

Each value should be a real number.

Default:

[0.0, 1.0]

E3-1-1-4 ISSUE\FA\required_step_size

The required quantization step size of the output for function approximation issues.
Acceptable format:
A positive real number.
Default:
0.01

E3-1-2 ISSUE\input_number

The number of dimensions of the issue (that is, the number of SDNN input dimensions).
Acceptable format:
An integer greater than or equal to 2.
Default:
2

E3-2 SDNN Parameters

E3-2-1 SDNN\PC

E3-2-1-1 SDNN\PC\n

The number of code pattern elements used in pattern coding.
Acceptable:
A positive even integer.
Default:
128

E3-2-1-2 SDNN\PC\random_seed

The random seed used for code pattern creation.
Acceptable:

- a vector of integers in the range [0, 4294967295];
- the character string hardware_entropy.

If hardware_entropy is set, the application will apply 10 random numbers with hardware-originated entropy as the random seed. The application uses MT19937 as its pseudo-random number generator.
Default:
hardware_entropy

E3-2-1-3 SDNN\PC\input_type_and_creation_method

The type (numerical or symbolic) and code pattern creation method for each input signal.
Acceptable:
A vector of character strings in the form [setting_for_input1, setting_for_input_2,...] (note that the number of elements must be the same as the dimensionality of the issue).

Each setting can be one of the following string types:

- NUMERICAL(RANDOM_INVERSE(*q*,*r*))
  Type: numerical input
  Creation: random-inverse method
  Arguments:
  - *q*: Number of input values. (The inputs values are quantized into *q* bins, with each bin assigned one code pattern.)
  - *r*: Number of different elements between neighboring code patterns.

- SYMBOLIC(RANDOM_INVERSE(*q*,*r*))
  Type: symbolic input
  Creation: random-inverse method
  Arguments:
  - *q*: Number of input symbols.
  - *r*: Number of different elements between neighboring code patterns.
- NUMERICAL(CORRELATION_MATRIX(
  correlation_matrix_file, batch_n, max_iteration, precision))
  Type: numerical input
  Creation: correlation-matrix method (this method produces code patterns according to a correlation-matrix file; see also Chapter E4-1)
  Arguments:
  - *correlation_matrix_file*: Name of the correlation-matrix file used for input.
  - *batch_n*: Number of elements of each sub-code pattern.
  - *max_iteration*: Maximum number of iterations for searching code patterns.
  - *precision*: The application repeats the code pattern searching process until the root-mean-square error between the ideal pattern and actual pattern correlations among the created code patterns is less than *precision*.
- SYMBOLIC(CORRELATION_MATRIX(
  correlation_matrix_file, batch_n, max_iteration, precision))
  Type: symbolic input
  Creation: correlation-matrix method
  Arguments:
  - *correlation_matrix_file*: Name of the correlation-matrix file used for input.
  - *batch_n*: Number of elements of each sub-code pattern.
  - *max_iteration*: Maximum number of iterations for searching code patterns.
  - *precision*: The application repeats the code pattern searching process until the root-mean-square error between the ideal pattern and actual pattern correlations among the created code patterns is less than *precision*.

Default:
Not defined.

E3-2-2 SDNN\SD

E3-2-2-1 SDNN\SD\combination_setting

This determines the method used to set the input combination for selective desensitization.
Acceptable:

- mutual
  Selective desensitization is conducted for all pairs of inputs.
- file
  Selective desensitization conducted for only those input pairs specified by the selective desensitization file (see also Chapter E4-2).

Default:
mutual

If you choose file, the following parameter must be set:

E3-2-2-2 SDNN\SD\filename

The name of the selective desensitization file.
Acceptable:
The name of the existing selective desensitization file.
Default:
Not defined.

E3-2-3 SDNN\PP

E3-2-3-1 SDNN\PP\random_seed

The random seed used to initialize the synaptic weights of the parallel perceptron. Please refer to Chapter **E3-2-1-2 SDNN\PC\random_seed** for details.

Acceptable:

- a vector of integers in the range [0, 4294967295].
- the character string `hardware_entropy`.

Default:

`hardware_entropy`

E3-2-3-2 SDNN\PP\initial_weight_range

The initial value range of the synaptic weights in the parallel perceptron.

Acceptable:

A two-dimensional vector in the form `[minimum value, maximum value]`; the minimum/ maximum value must be an integer.

Default:

`[-5, 5]`

E3-3 Parameters for Application Settings

E3-3-1 APP\print_progression

Determines whether the degree of progress in the training process is displayed.

Acceptable:

- `Y`: yes.
- `N`: no.

Default:

`Y`

E3-3-2 APP\multi_thread

Determines whether or not to perform parallel processing (OpenMP) in training/testing SDNN.

Acceptable:

- `Y`: yes.
- `N`: no.

Default:

`N`

If you choose `Y`, the following parameter must be set:

E3-3-2-1 APP\thread_number

The number of threads to be used.

Acceptable:

An integer between 1 and the number of CPU threads; if an integer larger than the number of threads is set, the application uses all of the threads.

Default:

`2`

E3-3-3 APP\autosave_filename

The `bipl::sdnn::SDNN` class automatically saves the model (hyper-parameters and synaptic weights of the SDNN) as the SDNN-model file following execution of the `Train` function. This parameter is set as the name of the autosaved SDNN-model file

Acceptable:

Any name with a .bin extension can be used for the autosave file of the trained model.

Default:

autosave.bin

E4 Other files used for initializing the SDNN

E4-1 Correlation-matrix file

The correlation-matrix method creates code patterns using a correlation matrix whose elements represent the ideal correlation coefficients between input values/symbols (all elements must be in the range [-1, 1]; see **Selective Desensitization Neural Networks** for details). The correlation-matrix file in this library can be used to set the correlation matrix. In this section, we explain how to set the parameters of the correlation-matrix file.

The correlation-matrix file must have a .csv extension and be written in a code that uses ASCII characters. Each row of the matrix in is given on a separate line with each element separated by a "," as shown in Figure E-2.

$1,\ c_{1,2},\ \cdots,\ c_{1,q}$

$c_{2,1},\ 1,\ \cdots,\ c_{2,q}$

$\vdots$

$c_{q,1},\ c_{q,2},\ \cdots,\ 1$

Figure E-2: Example of correlation-matrix file
for q input values/symbols

Note that inputting very large numbers of values (q) leads to exponentially increasing computational loads for preparing the code patterns; we recommend setting q to 100 or less.

E4-2 Selective desensitization file

Using a selective desensitization file, you can specify the pairs of inputs for which selective desensitization is conducted. This file must have a .csv extension and be written in a code using ASCII characters. Each input pair is described on a separate line in the form modified_input_number, modifier_input_number, as shown in Figure E-3. Please note that the input numbers must be specified with integers starting from 0. In this example, 0 represents input 1, 1 represents input 2, and 2 represents input 3.

```
0, 1
1, 0
0, 2
2, 1
```

Figure E-3: Example of selective desensitization file
with three inputs