Mex Usage

C1 Introduction

This document explains the use of the mex files libsdnn_train.mexw64 and libsdnn_test.mexw64 in the LIBSDNN/mex folder. These mex files are written in C++ and compiled using Visual Studio 2017.

C2 Installation

Move the both of the libsdnn_train.mexw64 and libsdnn_test.mexw64 into the Matlab(R) path folder. These .mexw64 files are produced by compiling, libsdnn_train.cpp and libsdnn_test.cpp on Matlab using the mex command (note that these .cpp files can only be compiled using the Visual C++ compiler). You can find additional information on the mex command in the Matlab article referenced at the end of this document (Matlab 2017).

C3 Quick Guide

1. Save the parameter files, including the selective desensitization file and/or correlation matrix file, to the current folder.
2. Use the libsdnn_train command to train the SDNN using training samples and produce an SDNN-model file.
3. Use the libsdnn_test command to estimate/recognize testing samples.

C4 Command details

C4-1 libsdnn_train

```
libsdnn_train(
        sdnn_parameter_file,
        input_list,
        target_list,
        completion_condition,
        training_result_file
)
```

This command trains the SDNN and produces an SDNN-model file containing the parameters and synaptic weights of the SDNN.
Arguments:

- sdnn_parameter_file (Character string)
  The name of the parameter file; please refer to **How to describe a parameter file** for further details.
- input list (Matrix)
  The list of training sample input vectors (see also Chapter C5).
- target list (Vector)
  The list of training sample target values (see also Chapter C5).
  completion_condition (Character string)
  The training-completion condition, which is represented in the form string. The setting can be one of the following strings:
  - iteration(*n*)
    This tells the SDNN to repeat training procedure *n* times.
  - rmse(*p*,*m*)
    This tells the SDNN to repeat the training process until the root-mean-square error is less than *p*. If the SDNN cannot satisfy the condition after *m* iterations, the training

process terminates. This completion condition can only be applied to function approximation issues.

- accuracy(*a*,*m*)
  This tells the SDNN to repeat the training process until the classification accuracy becomes greater than *a*. If the SDNN cannot satisfy this condition after *m* iterations, the training process terminates. This completion condition can only be applied to pattern recognition issues.

- training_result_file (Character string)
  The name of the SDNN-model file; its extension should be .bin.

Output:
void

C4-2 libsdnn_test

```
libsdnn_test(
        SDNN_model_file,
        input_list
)
```

Loads the SDNN-model file and recognizes/estimates the testing samples.
Arguments:

- SDNN_model_file (Character string)
  The name of the SDNN-model file. The SDNN-model file can be created using the libsdnn_train command as well as other applications (static library, command line tools).
- input list (Matrix)
  The list of tested sample input vectors.

Output:
A list of recognition/estimation result output vectors.

C5 How to make an input/target list

The input list matrix **I** and the target list vector $\boldsymbol{t}$ of training/testing samples must be written in the following forms:

$$\mathbf{I} = (\boldsymbol{i}_1 \quad \cdots \quad \boldsymbol{i}_n) = \left(\begin{pmatrix} i_{11} \\ \vdots \\ i_{1m} \end{pmatrix} \quad \cdots \quad \begin{pmatrix} i_{n1} \\ \vdots \\ i_{nm} \end{pmatrix}\right),$$

$$\boldsymbol{t} = (t_1 \quad \cdots \quad t_n).$$

Each column of the input list matrix is assigned to an input vector of a sample. The ordering of elements in the input list matrix must match that in the target vector. Please note that numerical inputs must be normalized in the range of [0, 1] and symbolic inputs must be specified using integers starting from 0. The target vector must be specified using integers starting from 0 when pattern recognition issues are applied.

C6 Example

Here, we demonstrate construction of an .m file using a function approximation problem (Nonaka et al. 2011) as an example. They approximated a two-variable function using an SDNN as follows:

$$\mathbf{f}(x,y) = \begin{cases} \mathbf{1}\ ((x-\mathbf{0.5})^2 + (y-\mathbf{0.5})^2 \le \mathbf{0.04}) \\ \dfrac{\mathbf{1}+x}{\mathbf{2}}\mathbf{sin}^2\left(\mathbf{6}\pi\sqrt{x}y^2\right)(\mathbf{otherwise}) \end{cases} (x, y \in [\mathbf{0}, \mathbf{1}]) \qquad \textbf{(C-1)}$$

C6-1 Making parameter files

In this demonstration, we use parameter_file.txt, which is saved in LIBSDNN/mex/examples. Please refer to the document **How to describe parameter files** for further details.

C6-2 Preparing training/test samples

The training and testing samples are stored in training_sample.csv and testing_sample.csv, respectively. In these files, each sample is written on a single line in the form target_value, input_1, input_2.

The following shows an example of the source used to prepare samples:

```
x = csvread('training_sample.csv');
training_input = x(:,2:3);
training_target = x(:,1:1);
x = csvread('testing_sample.csv');
test_input = x(:,2:3);
test_target = x(:,1:1);
```

C6-3 Training SDNN

To obtain the trained SDNN-model file, the libsdnn_train command is used as follows. In this example, we have transposed training_input and training_target and converted them into a suitable form following the procedure in Chapter C5 above:

```
libsdnn_train ('parameter_file.txt', training_input', training_target', 'iteration(300)',
                'SDNN_model.bin');
```

C6-4 Approximating functions

To approximate the function, use the libsdnn_test command as follows:

```
result = libsdnn_test('SDNN_model.bin',test_input');
```

The recognition/estimation results will be stored in result in the form of a row vector.

Reference

MathWorks, "Build MEX function from C/C++ or Fortran source code," https://www.mathworks.com/help/matlab/ref/mex.html?searchHighlight=mex%20build&s_tid=doc_srchtitle (accessed online 2017/7/23)

K. Nonaka, F. Tanaka, and M. Morita. Empirical comparison of feedforward neural network on two-variable function approximation. *IEICE TRANSACTIONS on Information and Systems* (in Japanese), J94 (12): 2114-2125, 2011.