Espam-AI manual

Requirements:

- python 2.7
- java 8+
- DARTS

Installation:

See README.md at the root folder

Execution from command line (after the tool is configured):

1. cd <profject root directory>
2. cd bin
3. ./espam [options]

Execution from jar (after jar is made by make jar):

1. cd <jar directory>
2. java -jar espam.jar [options]

Input:

- Path to single DNN model in .onnx or .json format + options (see below) *or*
- Path to single CSDF model in .json format *or*
- Directory with several DNN models (not recomended for large DNN models)

Output:

- DNN model evaluation results in .json format printed to console *or/and*
- Files, generated from DNN model and intended for CSDF-models for CSDF-models processing tools such as Sesame/DARTS/SDF3

Possibilities:

- evaluation : Evaluation of DNN/CSDF model in terms of power/performance
- generation : Generation of files intended for CSDF-models processing tools such as Sesame/DARTS/SDF3\

Tool running progress:

(1) *Model(s) reading*: read input model(s)

(2) *Model(s) conversion*:

- conversion of input .onnx/json DNN model(s) to internal Network model(s) (initiated automatically after onnx/json model reading)
- conversion of internal Network model(s) to CSDF model(s): (initiated automatically, if evaluation flag if set or/and one or multiple *-csdf output files generation flags are set)

(3) *Output files generation [optional]:* generates for input model output files, according to set files generation options.

(4) *Model evaluation [optional]:* evaluates one or several input models in terms of power/performance, by means of the DARTS/SDF3 tool.
TODO direct interface to SDF3 tool is not presented for current moment

Command-line options

Command-line options that take arguments

option	abbr	arguments	example
General options			
--generate	-g	Path to input DNN model	--generate ./tests/lenet.json
--evaluate	-e	Path to input DNN model	--evaluate ./tests/lenet.json
Model representation options (by default --layer-based option is set)			
--layer-based	-lb	none	-lb
--neuron-based	-nb	none	-nb
--block-based	-bb	number of blocks	--bb 20
--split-step [optional], for -bb models only	none	Number of child nodes, obtainbed after one layer splitting	--split-step 4
Hardware specification options			
--time-spec	none	Path to operators times specification	--time-spec ./time_spec.json
--energy-spec		Path to model energy specification	--energy-spec.json ./energy_spec.json
Additional options			
--dot-w	none	Width of generated .dot images in pixels	--dot-w 7000

Command-line flags (The command-line options that are either present or not).

flag	abbr	description
General flags		
--help	-h	Printout help
--version	-v	Printout program version
--verbose	-V	Printout program progress information
--copyright	none	Printout copyright
Input model type flag (--in-dnn is set by default)		
--in-dnn	none	Input model is dnn model in .onnx or .json format
--in-csdf	none	Input model is csdf model in .json format
Files generation flags		
--json	none	Generate DNN graph in .json format
--dot	none	Generate DNN graph image in .dot format
--json-csdf	none	Generate CSDF graph in .json format for DARTS
--xml-csdf	none	Generate CSDF graph in .xml fomat for SDF3
--dot-csdf	none	Generate CSDF graph image in .dot format

--sesame	none	Generate code templates for Sesame
--multiple-models	-m	Process not a single model, but multiple models. In this case path after –generate or –evaluate general flag should be the directory with several models. *Note: not recomended, escpecially for large models*

Files generation examples

For more information about files generation see comments, output files.

Example 1

Command:

./espam --generate ./tests/json/bvlc_alexnet_LB.json --sesame -lb

Description:

Generate layer-based CSDF graph for ./tests/json/bvlc_alexnet_LB.json DNN model. Provide it with Sesame application.

Expected output: files folder
<bvlc_alexnet_LB>

-app //sesame application in layer-based mode

Example 2

Command:

./espam --generate ./tests/json/lenet_NB.json --sesame --dot --json-csdf --xml-csdf --dot-csdf -nb

Description:

Generate neuron-based CSDF graph for ./tests/json/lenet_NB.json DNN model. Provide it with Sesame application, .dot images of DNN and CSDF model, .json and .xml files of CSDF model.

Expected output: files folder
<lenet_NB>

-app //contains sesame application in -nb mode
-dot //contains lenet_NB.dot picture of DNN model
-sdfg //corresponding CSDF graph directory
    -dot //contains lenet_NB.dot picture CSDF model
    -json //contains lenet_NB.json CSDF model suitable for DARTS\
    -xml //contains lenet_NB.xml CSDF model suitable for SDF3

Example 3

Command:

./espam --generate ./tests/onnx/mnist.onnx --sesame --dot --json-csdf --xml-csdf --dot-csdf -bb -100 --split-step 4

Description:

Generate block-based CSDF graph for ./tests/onnx/mnist.onnx DNN model. Split model until it reaches <=100 blocks with –split-step = 4 (For more information about layers splitting see Cooments, block-based models). Provide it with Sesame application, .dot images of DNN and CSDF model, .json and .xml files of CSDF model.

Expected output: files folder

<CNTKGraph_NB>
-app //contains sesame application in -bb mode
-dot //contains CNTKGraph.dot picture of DNN model
-sdfg //corresponding CSDF graph directory
-dot //contains CNTKGraph.dot picture of CSDF model
-json //contains CNTKGraph.json CSDF model suitable for DARTS\
-xml //contains CNTKGraph xml CSDF model suitable for SDF3

Model evaluation examples

Example 1

Command:

./espam --evaluate ./tests/json/bvlc_alexnet_LB.json

Description:

Evaluate ./tests/json/bvlc_alexnet_LB.json DNN model in terms of power/performance in -lb mode, set by default.

Expected output:

```
[@svetlana-Latitude-7280:/home/svetlana/jar_for_Simon] $ java -jar protobuf_inte
rface.jar --evaluate ./tests/json/bvlc_alexnet_LB.json -lb
{ "id": 0, "execution_time": 1.2997074085E10, "energy": 2.70885261651, "memory":
 2.77690192E8, "processors": 0 }
```

Example 2

Command:

./espam --evaluate ./tests/json/lenet_NB.json -nb

Description:

Evaluate ./tests/json/lenet_NB.json DNN model in terms of power/performance in -nb mode.

Expected output:

```
{ "id": 0, "execution_time": 4361469.0, "energy": 1.93671545212, "memory": 27828
0.0, "processors": 0 }
```

Example 3

Command:

./espam --evaluate ./tests/onnx/mnist.onnx -bb -100 --split-step 4

Description:

Evaluate ./tests/onnx/mnist.onnx DNN model in terms of power/performance. Split model until it reaches <=100 blocks with –split-step = 4 (For more information about layers splitting see Cooments, block-based models).

Expected output:

```
{ "id": 0, "execution_time": 9932160.0, "energy": 1.47208333333, "memory": 83424
.0, "processors": 0 }
```

Comments

1. Output files

Output files are generated in output models directory (output_models in protobuf_interface.jar root by default). For every model created a directory, named after the model.

E.g. for DNN called “LeNet”, the LeNet directory will be created.

Inside the folder there are files generated in accordance with the file generation flags. For all possible files the folder will have the following structure

The files structure:
<Model_name>
-app //sesame application
-dot //DNN model in .dot format
-json //DNN model in .json format
-sdfg //corresponding CSDF graph directory
-dot //CSDF model in .dot format
-json //CSDF model in .json format for DARTS\
-xml //CSDF model in .xml format for SDF3

NOTE: if directory with this name already exist it will be overwritten!

2. Block-based models
Block-based model takes on input layer-based DNN model and transforms it according to the following algorithm:

While (number of blocks<expected && split_up flag = true):
Layer bottleneck = find_bottleneck(); // bottleneck node search is preformed by DARTS
if (bottleneck can be split up)
evaluate number of layers after bottleneck layer splitting;
if (number of layers after splitting>expected)
return;
Split bottleneck into –split-step child Layers;
Split all Dependent layers (nonlinear/maxpool ones), following the bottleneck layer
else
return;

As transformation is performed over DNN models,files (.json / .dot etc) generated from the model in -bb mode will be different from ones for input (layer-based) DNN model. If .json for block-based model is already generated, it can be reused without additional splitting in -lb mode.

3. Recomendations and possible issues.

3.1. It is not recommended to use --generation with too many flags and multiple-models processing (--evaluation or ---generation) for heavy models, especially in -nb mode

3.2. Tool is not accounted on huge neuron-based models. It may work really long or even fall with JavaHeapSpace error with large DNN models in -nb mode.

3.3. Tool may complain on ‘unsafe’ libraries such as graphviz, used for .dot files generation.

3.4. If tool is frozen, kill the corresponding command line process and try to use the same command with –verbose (v) option. It should be shown on which step (model reading, model conversion, model evaluation etc.) the problem occured. Most likely, it will be DARTS, that still have some problems with large modls evaluation. DARTS is also called for repetition vector calculation before Sesame files generation and might also halt on large models.

3.5. If CSDF model is sent on input (with flag –in-csdf) , no specific DNN features (such as weights and proper data formats) will be taken into account. Thus, for processing DNN models it is recomended to use DNN .json/.onnx files. JSON files might be even preferable due to their small size.