NonLinearSystemNeuralNetworkFMU.jl

March 6, 2023

Contents

Part I

Home

Chapter 1

NonLinearSystemNeuralNetworkFMU.jl

Generate Neural Networks to replace non-linear systems inside OpenModelica 2.0 FMUs.

1.1 Table of Contents

1.2 Overview

The package generates an FMU from a modelica file in 3 steps (+ 1 user step):

1. Find non-linear equation systems to replace.
   - Simulate and profile Modelica model with OpenModelica using OMJulia.jl.
   - Find slowest equations below given threshold.
   - Find depending variables specifying input and output for every non-linear equation system.
   - Find min-max ranges for input variables by analyzing the simulation results.
2. Generate training data.

- Generate 2.0 Model Exchange FMU with OpenModelica.
- Add C interface to evaluate single non-linear equation system without evaluating anything else.
- Re-compile FMU.
- Initialize FMU using FMI.jl.
- Generate training data for each equation system by calling new interface.

3. Train neural network.

   - Step performed by user.

4. Integrate neural network into FMU

   - Replace equations with neural network in generated C code.
   - Re-compile FMU.

1.3 Installation

Clone this repository to your machine and use the package manager Pkg to develop this package.

```
(@v1.7) pkg> dev /path/to/NonLinearSystemNeuralNetworkFMU
julia> using NonLinearSystemNeuralNetworkFMU
```

Part II

Main

Chapter 2

Main Data Generation Routine

To perform all needed steps for data generation the following functions have to be executed:

1. `profiling`
2. `generateFMU`
3. `addEqInterface2FMU`
4. `generateTrainingData`

These functionalities are bundled in `main`.

2.1 Functions

`NonLinearSystemNeuralNetworkFMU.main` – Function.

```
main(modelName, moFiles;  options=OMOptions(workingDir=joinpath(pwd(), modelName)),
↪  reuseArtifacts=false, N=1000)
```

Main routine to generate training data from Modelica file(s). Generate BSON artifacts and FMUs for each step. Artifacts can be re-used when restarting main routine to skip already performed stepps.

Will perform profiling, min-max value compilation, FMU generation and data generation for all non-linear equation systems of `modelName`.

Arguments

- `modelName::String`: Name of Modelica model to simulate.
- `moFiles::Array{String}`: Path to .mo file(s).

Keywords

- `options::OMOptions`: Settings for OpenModelcia compiler.
- `reuseArtifacts=false`: Use artifacts to skip already performed steps if true.
- `N=1000::Integer`: Number of data points fto genreate or each non-linear equation system.

Returns

- `csvFiles::Array{String}`: Array of generate CSV files with training data.
- `fmu::String`: Path to unmodified 2.0 ME FMU.
- `profilingInfo::Array{ProfilingInfo}`: Array of profiling information for each non-linear equation system.

See also `profiling`, `minMaxValuesReSim`, `generateFMU`, `addEqInterface2FMU`, `generateTrainingData`.

`source`

Part III

Profiling

Chapter 3

Profiling Modelica Models

3.1 Functions

`NonLinearSystemNeuralNetworkFMU.profiling` – Function.

```
profiling(modelName, moFiles; pathToOmc, workingDir, threshold = 0.03)
```

Find equations of Modelica model that are slower then threashold.

Arguments

- `modelName::String`: Name of the Modelica model.
- `moFiles::Array{String}`: Path to the *.mo file(s) containing the model.

Keywords

- `options::OMOptions`: Options for OpenModelica compiler.
- `threshold=0.01`: Slowest equations that need more then `threshold` of total simulation time.
- `ignoreInit::Bool=true`: Ignore equations from initialization system if `true`.

Returns

- `profilingInfo::Vector{ProfilingInfo}`: Profiling information with non-linear equation systems slower than `threshold`.

source

`NonLinearSystemNeuralNetworkFMU.minMaxValuesReSim` – Function.

```
minMaxValuesReSim(vars, modelName, moFiles; pathToOmc="" workingDir=pwd())
```

(Re-)simulate Modelica model and find miminum and maximum value each variable has during simulation.

Arguments

- `vars::Array{String}`: Array of variables to get min-max values for.
- `modelName::String`: Name of Modelica model to simulate.
- `moFiles::Array{String}`: Path to .mo file(s).

Keywords

- `pathToOmc::String=""`: Path to OpenModelica Compiler omc.
- `workingDir::String=pwd()`: Working directory for omc. Defaults to the current directory.

Returns

- `min::Array{Float64}`: Minimum values for each variable listed in `vars`, minus some small epsilon.
- `max::Array{Float64}`: Maximum values for each variable listed in `vars`, plus some small epsilon.

See also `profiling`.

source

3.2 Structures

`NonLinearSystemNeuralNetworkFMU.ProfilingInfo` – Type.

```
ProfilingInfo <: Any
```

Profiling information for single non-linear equation.

- `eqInfo::EqInfo`: Non-linear equation
- `iterationVariables::Array{String}`: Iteration (output) variables of non-linear system
- `innerEquations::Array{Int64}`: Inner (torn) equations of non-linear system.
- `usingVars::Array{String}`: Used (input) variables of non-linear system.
- `boundary::NonLinearSystemNeuralNetworkFMU.MinMaxBoundaryValues{Float64}`: Minimum and maximum boundary values of `usingVars`.

source

`NonLinearSystemNeuralNetworkFMU.EqInfo` – Type.

```
EqInfo <: Any
```

Equation info struct.

- `id::Int64`: Unique equation id
- `ncall::Int64`: Number of calls during simulation
- `time::Float64`: Total time [s] spend on evaluating this equation.
- `maxTime::Float64`: Maximum time [s] needed for single evaluation of equation.
- `fraction::Float64`: Fraction of total simulation time spend on evaluating this equation.

source

3.3 Examples

Find Slowest Non-linear Equation Systems

We have a Modelica model `SimpleLoop`, see test/simpleLoop.mo with some non-linear equation system

$$
\begin{aligned}
r^2 &= x^2 + y^2 \\
rs &= x + y
\end{aligned}
$$

We want to see how much simulation time is spend solving this equation. So let's start `profiling`:

```
julia> using NonLinearSystemNeuralNetworkFMU


julia> modelName = "simpleLoop";


julia> moFiles = [joinpath("test","simpleLoop.mo")];



julia> profilingInfo = profiling(modelName, moFiles, omc; threshold=0)
ERROR: MethodError: no method matching profiling(::String, ::Vector{String}, ::String; threshold=0)
Closest candidates are:
  profiling(::String, ::Array{String}; options, threshold, ignoreInit) at
↪  ~/work/NonLinearSystemNeuralNetworkFMU.jl/NonLinearSystemNeuralNetworkFMU.jl/src/profiling.jl:332
```

We can see that non-linear equation system 14 is using variables s and r as input and has iteration variable y. x will be computed in the inner equation.

```
julia> profilingInfo[1].usingVars
ERROR: UndefVarError: profilingInfo not defined

julia> profilingInfo[1].iterationVariables
ERROR: UndefVarError: profilingInfo not defined
```

So we can see, that equations 14 is the slowest non-linear equation system. It is called 2512 times and needs around 15% of the total simulation time, in this case that is around 592 μs.

During `profiling` function `minMaxValuesReSim` is called to re-simulate the Modelica model and read the simulation results to find the smallest and largest values for each given variable.

We can check them by looking into

```
julia> profilingInfo[1].boundary.min
ERROR: UndefVarError: profilingInfo not defined

julia> profilingInfo[1].boundary.min
ERROR: UndefVarError: profilingInfo not defined
```

Part IV

Data Generation

Chapter 4

Training Data Generation

To generate training data for the slowest non-linear equations found during Profiling Modelica Models we now simulate the equations multiple time and save in- and outputs.

We will use the Functional Mock-up Interface (FMI) standard to generate FMU that we extend with some function to evaluate single equations without the need to simulate the rest of the model.

4.1 Functions

`NonLinearSystemNeuralNetworkFMU.generateFMU` – Function.

```
generateFMU(modelName, moFiles; [pathToOmc], workingDir=pwd(), clean=false)
```

Generate 2.0 Model Exchange FMU for Modelica model using OMJulia.

Arguments

- `modelName::String`: Name of the Modelica model.
- `moFiles::Array{String}`: Path to the *.mo file(s) containing the model.

Keywords

- `options::OMOptions`: Options for OpenModelica compiler.

Returns

- Path to generated FMU `workingDir/<modelName>.fmu`.

See also `addEqInterface2FMU`, `generateTrainingData`.

source

`NonLinearSystemNeuralNetworkFMU.addEqInterface2FMU` – Function.

```
addEqInterface2FMU(modelName, pathToFmu, eqIndices; workingDir=pwd())
```

Create extendedFMU with special_interface to evalaute single equations.

Arguments

- `modelName::String`: Name of Modelica model to export as FMU.

- `pathToFmu::String`: Path to FMU to extend.
- `eqIndices::Array{Int64}`: Array with equation indices to add equiation interface for.

Keywords

- `workingDir::String=pwd()`: Working directory. Defaults to current working directory.

Returns

- Path to generated FMU `workingDir/<modelName>.interface.fmu`.

See also `profiling`, `generateFMU`, `generateTrainingData`.

source

`NonLinearSystemNeuralNetworkFMU.generateTrainingData` – Function.

```
generateTrainingData(fmuPath, workDir, fname, eqId, inputVars, min max, outputVars;
                     N=1000, nBatches=1, append=false)
```

Generate training data for given equation of FMU.

Generate random inputs between `min` and `max`, evalaute equation and compute output. All input-output pairs are saved in `fname`.

Arguments

- `fmuPath::String`: Path to FMU.
- `workDir::String`: Working directory for generateTrainingData.
- `fname::String`: File name to save training data to.
- `eqId::Int64`: Index of equation to generate training data for.
- `inputVars::Array{String}`: Array with names of input variables.
- `min::AbstractVector{T}`: Array with minimum value for each input variable.
- `max::AbstractVector{T}`: Array with maximum value for each input variable.
- `outputVars::Array{String}`: Array with names of output variables.

Keywords

- `N::Integer = 1000`: Number of input-output pairs to generate.
- `nBatches::Integer = 1`: Number of batches to separate `N` into to generate data in parallel.
- `append::Bool=false`: Append to existing CSV file `fname` if true.

See also `generateFMU`, `generateFMU`.

source

4.2 Examples

First we need to create a Model-Exchange 2.0 FMU with OpenModelica.

This can be done directly from OpenModelica or with `generateFMU`:

```
using NonLinearSystemNeuralNetworkFMU #hide
omc = string(strip(read(`which omc`, String))) #hide

fmu = generateFMU("simpleLoop",
                  ["test/simpleLoop.mo"];
                  pathToOmc = omc,
                  workingDir = "tempDir")
rm("tempDir", recursive=true, force=true) #hide
```

Next we need to add non-standard C function

```
fmi2Status myfmi2evaluateEq(fmi2Component c, const size_t eqNumber)
```

that will call <modelname>_eqFunction_<eqIndex>`(DATA* data, threadData_t *threadData)` for all non-linear equations we want to generate data for.

Using `addEqInterface2FMU` this C code will be generated and added to the FMU.

```
interfaceFmu = addEqInterface2FMU("simpleLoop",
                                  fmu,
                                  [14],
                                  workingDir = "tempDir")
rm("tempDir", recursive=true, force=true) #hide
```

Now we can create evaluate equation 14 for random values and save the outputs to generate training data.

```
using CSV
using DataFrames
generateTrainingData(interfaceFmu,
                     "simpleLoop_data.csv",
                     14,
                     ["s", "r"],
                     [0.0, 0.95],
                     [1.5, 3.15],
                     ["y"];
                     N = 10)
df =  CSV.File("simpleLoop_data.csv")
rm("simpleLoop_data.csv", force=true) #hide
```