Data driven Computational Mechanics at EXascale

Data driven Computational Mechanics at EXascale

Work program topic: EuroHPC-01-2019
Type of action: Research and Innovation Action (RIA)

REPORT ON THE DCoMEX-MAT APPLICATION

DELIVERABLE D7.3 (software module)

Version No 1

http://www.dcomex.eu/
This project has received funding from the European High-Performance Computing Joint Undertaking Joint Undertaking ('the JU'), under Grant Agreement No 956201

DOCUMENT SUMMARY INFORMATION

Project Title	**Data driven Computational Mechanics at EXascale**
Project Acronym	DCoMEX
Project No:	956201
Call Identifier:	EuroHPC-01-2019
Project Start Date	01/04/2021
Related work package	WP 7
Related task(s)	Task 7.2
Lead Organisation	UCY
Submission date	18/02/2021
Re-submission date	
Dissemination Level	PU

Quality Control:

	Who	Affiliation	Date
Checked by internal reviewer	George Stavroulakis	NTUA	15/02/2024
Checked by WP Leader	T. Stylianopoulos	UCY	15/02/2024
Checked by Project	Vissarion Papadopoulos	NTUA	15/02/2024

Document Change History:

Version	Date	Author (s)	Affiliation	Comment

Contents

1. Description

In **DELIVERABLE 7.3 (software module)** that is associated with **WP2 "Surrogate modelling"**of the DCoMEX project, a report is provided regarding the software implementations done in MSolve for the second application (**DCoMEX-MAT)** of the DCoMEX project (for a detailed description of the theoretical aspect of this application see the supplementary report in **DELIVERABLE D7.4 (theory manual)**).

The development of the algorithmic procedure in MSolve for the second application consists of 2 independent parts inside the MSolve environment:

The first part includes the data collection for the training of the feed forward neural networks (FFNNs). For that, an amount of finite element boundary value problems has to be solved for different representative volume elements (RVEs) of various length scales (e.g. a CNT-reinforced cement paste, a CNT-reinforced mortar, a CNT-reinforced concrete). After the finite element solutions, the data can be stored and used at a future time as training samples in any machine learning framework.

The second part involves the FFNN training with the gathered data samples. An external machine learning library, based on Python's TensorFlow, is utilized by integrating it in MSolve with the proper customization. For the training, the architecture of the FFNN and other hyperparameters can be freely chosen. The trained FFNNs can then be used as material models for future FEM analyses. To do that, a tailored constitutive law is developed which can predict stress responses and tangent moduli based on the FFNN predictions for given strains.

All code can be found in https://github.com/mgroupntua/Constitutive, specifically in the repository that is named MGroup.Constitutive.Structural.MachineLearning

2. Algorithmic implementation in the MSolve software

To be able to develop FFNNs in MSolve the `FeedForwardNeuralNetwork` class has been developed, based on the Tensorflow library, and is located in: https://github.com/mgroupntua/MachineLearning/tree/develop/src/MGroup.MachineLearning.TensorFlow/NeuralNetworks

NeuralNetworkMaterial3D Constructor

```
FeedForwardNeuralNetwork(normalizationX, normalizationY, optimizer, lossFunc,
neuralNetworkLayer, epochs, batchSize, seed, classification)

normalizationX: type of normalization used in the input data

normalizationY: type of normalization used in the output data

lossFunc: type of loss function

neuralNetworkLayer: list of layers of specific neuron size and activation function

epochs: total iterations of a forward and backward pass through the FFNN during training

batchSize: number of samples in one forward and backward pass through the FFNN during
training

seed: seed of the pseudo-random number generator

classification: Boolean option for either a regression or classification problem
(default=false)
```

NeuralNetworkMaterial3D Methods

```
Train(stimuli, responses)

stimuli: training input data

responses: training output data


method objective: trains the FFNN with the provided input and output data
```

```
EvaluateResponses(stimuli)

stimuli: input data


method objective: provides the prediction of the output of the trained FFNN for specific
input data
```

```
EvaluateResponseGradients(stimuli)

stimuli: input data


method objective: provides the prediction of the gradient of the output with respect to
the input of the trained FFNN for specific input data
```

A special constitutive law has been created which reads a pre-trained FFNN and then can predict stress vector responses and tangent constitutive matrices based on the FFNN predictions.

NeuralNetworkMaterial3D Constructor

```
NeuralNetworkMaterial3D(neuralNetwork, materialParameters)

neuralNetwork: pre-trained FFNN on specific strain/parameter input states and stress
output states

materialParameters: material parameters according to which the FFNN has be trained
(optional)
```

NeuralNetworkMaterial3D Methods

```
UpdateConsitutiveMatrixAndEvaluteResponse(strainsIncrement)

strainsIncrement: the incremental strain vector on a specific gauss point for a specific
step of the iterative analysis


method objective: updates the local variables of the NeuralNetworkMaterial3D object
```

```
GetConstitutiveMatrix()


method objective: calculates the tangent constitutive matrix for the current total strain
by using the pre-trained FFNN
```

```
CalculateNextStressStrainPoint()


method objective: calculates the total stress vector for the current total strain by using
the pre-trained FFNN
```

The main algorithmic procedure takes place in the class called **NeuralNetworkMaterialBuilder.** This class includes both steps required for the development of the hierarchy of the FFNNs, namely the solution of the boundary value problems imposed by each scale RVEs for the data collection, and the training of the FFNNs with the collected data after choosing their parametrization. These steps can be performed independently by two methods inside **NeuralNetworkMaterialBuilder.**

NeuralNetworkMaterialBuilder Methods

```
GenerateStrainStressData()

method objective: this is the main script for the solution of the finite element problems
as described by the RVEs and the subsequent data collection. In this script the user has
the option to select the type of the RVE for the analysis (i.e. the different phases of the
microstructure, the material of each phase, the morphological structure). Additionally, the
strain sampling ranges and the sampling distributions can be chosen, which eventually affects
the random sampling during the RVE problem solutions. The user assigns the total desired
number of input-output pairs. Until the total number of RVE solutions is reached, after each
solution, the strain input - stress output data are stored in a user specified path in the
system environment. These data can be used for the training of any machine learning algorithm
either in MSolve or in any other external software (e.g. Python).
```

```
TrainNeuralNetwork()

method objective: this is the main script for the training of the FFNN with already collected
training data. In this script the user has the option to select FFNN parameters such as the
number of hidden layers, the neurons for each hidden layer, the activation function for each
hidden layer, the optimization algorithm, the loss function, the epochs and batches. The
data are loaded from the user specified path in the system environment which were previously
stored. The FFNN is trained by employing the FeedForwardNeuralNetwork class, and then again,
in a user specified path three files are saved, namely the netPathFile(contains information
about the architecture of the FFNN), the weightsPathFile(contains information about the
trained weights and biases of the FFNN) and the normalizationPathFile(contains information
about the type of normalization used during the training process). These files can then be
loaded at the initialization of an object of a NeuralNetworkMaterial3D class during a FEM
analysis.
```