Training a neural network to replace “dataset projections” in d-refinement

This notebook is a companion to “Mesh d-refinement: a data-based computational framework to account for complex material response”.
The neural network referred to in the appendix is devised herein.

J. Garcia-Suarez, 2023

Prepare data for training

Dataset

We use exactly the same data that was used for DDCM d-refinement (section 4), these data were obtained from sampling the octet-truss unit cell (section 3.2.).

```
numLoadSteps = 29;(*number of steps per trajectory*)
```

```
SetDirectory[NotebookDirectory[]];
dataNL = Import["dataset.csv"];(*make sure that the file
 "dataset.csv" is in the same directory as the notebook, or add path*)
```

Save per-element stress-strain evolution:

```
numElements = Dimensions[dataNL][[1]] / numLoadSteps;
```

Characteristic values

These are necessary to normalize the values the NN handles, as it will be much more efficient if there are no units and everything is O(1)

```
Ec = 3591851.;
σc = 16.*10^4;(*pressure*)
ϵc = σc / Ec;
```

Save trajectories

Prepare data for the NN: reshape and add the origin for each trajectory

```
elementTrajectories = ConstantArray[, {numElements}];
Do[
 data = dataNL[[1 + (ii - 1) * numLoadSteps ;; numLoadSteps * ii, All]];
 data = PrependTo[data, {0., 0., 0., 0., 0., 0.}];
 elementTrajectories[[ii]] = data;
 , {ii, 1, numElements}]
```

Separate strain and stress, and normalize

```
allStrain = 1/εc * (Take[#, 3] & /@ Flatten[elementTrajectories, 1]);
allStress = 1/σc * (Drop[#, 3] & /@ Flatten[elementTrajectories, 1]);
```

Put data in correct input form and separate batches (training validation and verification):

```
nTraining = Floor[0.7 * numElements];(*70% data training*)
nValidation = Floor[0.9 * numElements] - nTraining;
(*10% to run validation during training*)
nVerification = numElements - (nTraining + nValidation);
(*last 10% to test after training*)
```

```
trainingData = Table[allStrain[[ii]] → allStress[[ii]], {ii, 1, nTraining}];
testData = Table[allStrain[[ii]] → allStress[[ii]], {ii, 1 + nTraining, numElements}];
```

Neural Network

Define architecture

This simple architecture is made by:

- Input (3 neurons, one per strain component -- ϵ_{xx}, ϵ_{yy} and γ_{xy})
- Linear layer (6 neurons)
- Non-linear ReLu layer (6 neurons)
- Linear layer (18 neurons)
- Non-linear ReLu layer (18 neurons)
- Linear layer (6 neurons)
- Output (linear layer w/ 3 neurons, one per stress component -- σ_{xx}, σ_{yy} and τ_{xy})

```
net = NetChain[{6, Ramp, 18, Ramp, 6, 3}, "Input" → 3];
```

Initialize weights

```
initializedNet = NetInitialize[net, Method → "Xavier"];
```

Training

```
trainedNet =
 NetTrain[initializedNet, trainingData, ValidationSet → testData, Method → "ADAM"]
```

Out[ ]= NetChain[

	Input	vector (size: 3)
1	LinearLayer	vector (size: 6)
2	Ramp	vector (size: 6)
3	LinearLayer	vector (size: 18)
4	Ramp	vector (size: 18)
5	LinearLayer	vector (size: 6)
6	LinearLayer	vector (size: 3)
	Output	vector (size: 3)

]

Final test

Compute the verification error (%):

```
verAvError = 100/nVerification *
  Total@Table[Norm[σc * trainedNet /@ (elementTrajectories[[ii, All, 1 ;; 3]]/ϵc) -
       elementTrajectories[[ii, All, 4 ;; 6]]] / Norm[elementTrajectories[[
       ii, All, 4 ;; 6]]], {ii, numElements - nVerification, numElements}]
```

Out[ ]= 0.959105

Less than 1%