**Neural network documentation**

Use \*Linux\* platform. Ubuntu, Fedora is preferred.

One need to install the following python packages in order to work properly:

conda, netCDF4, numpy, matplotlib, random, csv, reader, torch, tqdm, h5py, and glob.

Variables definition:

* **time**: represents the simulation time in CPU time units. To get the corresponding real time, just divide it by 24×60×60.
* **alpha\_ppt**: represents the fraction of alpha precipitate formed at a given time in the system.
* **beta\_ppt**: represents the fraction of beta precipitate formed at a given time in the system.
* **te**: represents total energy density of the system at a particular time.
* **x\_alpha**: represents the composition value of one of the common tangent (corresponding to alpha phase) to the spinodal decomposition curve.
* **x\_beta**: represents the composition value of the other common tangent (corresponding to the beta phase) to the spinodal decomposition curve.

Alpha and Beta phase is defined from the perspective of the user.

This model is developed using PyTorch platform. Please see the documentation and tutorials for using PyTorch especially \*dataloader\*, \*saving models\*, and \*loading saved models\*.

This neural network model is built in 3 stages:

1. **Initial data preparation.**
2. **Network training, and testing.**
3. **Making predictions and post analysis of results.**

**1. Initial data preparation**

In this section, we need the .csv and .e file generated from the MOOSE simulation. Suppose you have these 2 files, then put these files in a folder named **initialPrep**. Also, you need the python script **timeDependentExodusConverter.py** inside the **initialPrep** folder. Delete the first line from the \*.csv\* file as it represents header which is strings, and we need numbers. If the first line from the \*.csv\* file is not deleted, the user has to adjust the **timeDependentExodusConverter.py** file in such a way that the script will not read the first line otherwise it will throw an error at later stages.

In the script **timeDependentExodusConverter.py** script, carefully examine the list index of 6 parameters i.e., time, alpha\_ppt, beta\_ppt, te, x\_alpha, and x\_beta in line 24-29. For example, line 24: b[0] represents the \*time\* as the user can verify from the \*.csv\* file that the 1st column in that \*.csv\* file represents time. Further, provide a name for the output file in line 38 (initially it is set to compiled\_MoV.txt).

Finally, provide the time interval to pick the image from the \*.e\* file and its 6 parameters from the \*.csv\* file. This is done because MOOSE generates a lot of images at different time steps which can pile up to a huge dataset. Taking files at different time intervals reduce the data burden and also provides well separated microstructures. For example, if MOOSE generates 10,000 microstructures of a system, then we should take 500 images from it by specifying the interval \*20\* in line 48 in the file **timeDependentExodusConverter.py**.

Run the **timeDependentExodusConveter.py** script by typing following command:

* python timeDependentExodusConverter.py

Make sure you have python installed and the path variable/environment is set correctly otherwise it will throw errors. It will generate a \*.txt\* file. Suppose name of the generated file is **compiled\_MoV.txt**, then its contents are as follows:

* Each row represents a microstructure at a particular time step. Taking the above example, you should have 500 lines/rows in the **compiled\_MoV.txt** file.
* 1st column: represents time.
* 2nd column: represents alpha\_ppt.
* 3rd column: represents beta\_ppt.
* 4th column: represents te.
* 5th column: represents x\_alpha.
* 6th column: represents x\_beta.
* From 7th column to rest of the columns: represents microstructure.

We consider a 128×128 microstructure, so we should have 16384 columns after the 6th column. So total number of columns will be 16390.

Ones you have the **compiled\_MoV.txt** file ready, copy that file in another folder named **data**. You probably again delete the 1st line in the **compiled\_MoV.txt** file because it corresponds to the microstructure at t = 0 which is nothing but distributed random noise.

In the **data** folder, put the **mycfdata.py** file and run it by using the following command:

* python mycfdata.py

This will normalize your data in the range [-1.0, 1.0] and will generate a series of \*t\_\*.txt\* files which is equal to the number of rows in your **compiled\_MoV.txt** file. Each of these \*t\_\*.txt\* file represents the system’s state with its first 6 values denotes the parameters defined above followed by 16384 values defining the microstructure of the system.

Again, in the same **data** folder, put the **myDataGenerator.py** file. Change line 4 of the **myDataGenerator.py** file and provide the correct path of your **data** folder. Save and run it by using the following command:

* python myDataGenerator.py

This will do a random 75% and 25% partition of all the 500 data points or microstructures (from above example) for training and testing respectively. Further, this will create 2 files: **dataForTraining.txt**, and **dataForTesting.txt** which will be used to train the neural network. These 2 files contains the list of addresses of different \*t\_\*.txt\* files generated as a result of executing the **mycfdata.py** python script.

Now, we will have the **compiled\_MoV.txt** file’s address also written in one of the 2 generated files. We don’t want the address of **compiled\_MoV.txt** file and thus, we will search for **compiled\_MoV.txt** file in these 2 files viz., **dataForTraining.txt** and **dataForTesting.txt** and delete the line which contains the address of the **compiled\_MoV.txt** file.

Finally, copy your **dataForTraining.txt** and **dataForTesting.txt** file in another folder named **network**.

**2. Network training, and testing**

For this, go into the **network** folder. It has **dataForTraining.txt** and **dataForTesting.txt** file from previous steps. Also, put the **cnnNetwork.py** file into this folder.

Main execution starts from line 142 of the file **cnnNetwork.py**. Before executing this script, we need to take care of the following things:

* Line 146: specify the training dataset file name i.e., **dataForTraining.txt**.
* Line 147: specify the testing dataset file name i.e., **dataForTesting.txt**.
* Line 150: specify the batch size. If user is not sure what to choose, the maximum number of datapoints (500 from the above example) can be specified.
* Line 165: specify the learning rate. Default is 0.001 but the user can play with this value to find an optimum learning rate. If not sure, then specify 0.001.
* Line 180: specify the number of training cycles for the neural network.
* Line 206: specify the interval at which the neural network’s state has to be saved in order to use it at a later time to make predictions.
* Line 208, and 210: carefully specify the location/path where the network and optimizer has to be saved.
* Line 17: here 6 represents your 6 parameters explained above.
* Line 18: here 16384 represents the number of pixels in your microstructure. 128×128 is 16384.
* Line 45: here we are reshaping 16384 pixels into a square 128×128 matrix to actually represents a microstructure.

After taking care of the above points, run the script by doing the following:

* python cnnNetwork.py

Executing above command will start your neural network training. It will calculate the training the test loss curves which can be plotted by executing the following command:

* python plotresults.py

This will also save your neural network models and its corresponding optimizers at the intervals specified in line 206 of the **cnnNetwork.py** file.

**3. Making predictions and post analysis of results**

We will use the **cnnPrediction.py** script to make predictions from the trained neural network. **cnnPrediction.py** is basically same as the **cnnNetwork.py** file except it has no training class. It just has testing class in it.

Before running the **cnnPrediction.py** script, make the following changes to it:

* Line 124: specify the filename for which prediction has to be done. Generally, this file is same as **dataForTraining.txt** or **dataForTesting.txt** file but with just 1 row/line. This row/line is the address of the microstructure file for which prediction has to be made. An elegant way to do this is to copy the **dataForTesting.txt** as rename it as **roughTesting.txt** and then delete all the lines except the 1 line for which prediction has to be done.
* Line 141: Specify the saved model to be used. Generally, the last model or the best model (which has the lowest training and testing loss value) has to be used for prediction. After training, you’ll have a series of **Network-\*** files saved in your **network** directory. So, specify the last **Network-\*** name in this line.

With the above changes, run this file using the following command:

* python cnnPrediction.py

This will generate the **prediction.txt** file which is the neural network prediction of the microstructure specified in the **roughTesting.txt** file. This **prediction.txt** file contains data in the same fashion as your **compiled\_MoV.txt** file.

Now, make another folder named **postAnalysis** and move your **prediction.txt** file, \*.e\* file, and \***imageDataPlotter.py**\* file in that folder. Execute the \***imageDataPlotter.py**\* file in order to get the neural network predicted microstructure.