**Investigation summary Golam Gause Jaman**

**Mentors: Fei Zhou & Saad Khairallah**

**Objective:**

Accelerating ALE3D-CAFE grain growths computations applied to Additive Manufacturing (AM) process.

**Approach:**

The major task is the development of deep learning model to learn and predict the simulations conducted in ALE3D integrated computational materials (ICM) tool that includes cellular Automata and Finite Element (CAFE) capability. As a baseline, Recurrent Neural Network (RNN) are considered with additional autoencoders. The network took few iterations of changes before testing ground is established.

**Data:**

The ground truth data is generated using the ICM tool and provided in an accessible directory as .dat files. Files are named in order, obeying a time sequence. The .dat files are read as tabular data to conduct initial inspection on different features and properties. Features considered are cell state, euler angles, normalized octahedron size, temperature and the temperature difference between consecutive time stamps. Features are grouped as a 3D time slices with <x,y,z> presenting coordinates of feature voxel. Scripts are written to visualize the 3D slices and understand the data across different times stamps. The data considered in the study are isotropic and anisotropic addressed as iso and aniso respectively.

**Pre-process:**

The data are processed to construct the time dependent stack of feature volumes used for the training and validation steps. In addition, euler angles are decoded to rotation matrix to aid the learning curve. Temperature difference feature is derived from consecutive slices and concatenated during the pre-process.

**Investigations:**

The network considered in the study is Predictive RNN or PredRNN with added autoencoders. The network hyperparameter domains are explored manually to setup a standard configuration. Hyperparameters tested include number of layers deep, learning rate, input time steps, output time steps, autoencoder and euler angle decoder. Performances are measured in terms of accumulating mean square error across all features, sequence and time steps. Time difference feature is excluded during the initial training runs. Highest performing network among all the configurations tested contains following setup:

* Number of layers: 3 (beyond 3, runs out of memory)
* Number of time steps input: 3-5
* Number of time steps output: 5
* Kernel size kept at 3
* Autoencoder connection is active
* Time difference feature is included if using aniso data
* Learning rate: 1e-02
* Euler angle to rotation matrix conversion applied

The network significantly improves when rotation matrix is included. Also, larger the output time steps further the network validates with lower error. However, the training process goes out of memory if the time steps for output is set to beyond 5. Autoencoder connection slightly improves the network. Time difference feature significantly improves the network but only applicable to aniso data as the temperature gradient varies with time (unlike iso data). Larger learning rate helps faster the convergence but too large could result in missing global optimal solution. Hence, more data/samples needed to confirm the best learning rate. At the time of this documentation best learning rate appear to be 1e-02. Deeper the network showed growing performance but anything beyond 3 as number of layer parameters ran out of memory. A script is written to aid the ranking process for identifying most effective network configuration.

Initial observations with 30 steps roll out shows exploding output features and characteristics disobeying the physics of the problem. The inferencing during validation is further enhanced by adding set of constrains that eliminates obvious mistakes. The hard rules applied until this document is created are as follows:

* Cell state 0 (inactive) should remain inactive in future steps.
* Octahedron size is not applicable during cell state 0 and 1(liquid) and presented as -1.
* Rotation matrix are also not applicable during cell state 0 and 1(liquid) and presented as -1 as no orientation unless mushy or solid (2,3 respectively).
* Octahedron size is maximum during mushy or solid state and presented as 1.
* Rotation matrix is inherited if transition happens among mushy and solid states.

The hard rules enhanced the previous best model further. However, training with the hard rules causes vanishing gradient flow and so hard rule should only be practiced during inference. The investigation is extended by analyzing confusion matrix of cell states and the confusion matrix of cell states given previous state. The confusion matrix analysis shows how mushy state is being the most challenging state to predict as it gets often confused with either liquid or solid by the highest performing network.

**Prospective work:**

The study is currently containing baseline work for the main objective. This work can be extended to analyze further by producing probability function for euler values given previous state and correctly predicted current state. Moreover, the work needs to be stretched to other promising network architecture such as Graphical Neural Network (GNN) that allows abstract neighbors for spatial-temporal learning.

**Work Environment:**

The investigations including pre-process work, running training jobs, validation and post analysis are done using modification of NPS module (written in python, provided by Fei Zhou). NPS module requires specific versions of python tools to work (please contact Fei Zhou for more information on the module). A virtual environment is set with python 3.8, NPS module setup and exporting NPS main folder to the .profile/.bashrc (one might want to export script folder under main NPS to the exporting path for convenient plotting). Modification of NPS module is recorded in the **difference.txt (read info.txt).** In addition, for visualization and post analysis, plot-training.sh, plot-hist.py and animate-3d.py are used. Jobs are written in bashscript to send training batch jobs. The work is conducted using Lassen either through pbatch or pdebug using “heas” as the resource bank.

**Example of pbatch/pdebug used during the work:**

* bsub -nnodes 1 -W 12:0 -q pbatch -G heas jobname.sh
* or interactive node for debugging using: lalloc 1 -W 2:0 -q pdebug -G heas

**Following link provides more information on LC/LSF commands:**

<https://hpc.llnl.gov/banks-jobs/running-jobs/lsf-quick-start-guide>

For building conda environment: <https://docs.conda.io/projects/conda/en/4.6.0/_downloads/52a95608c49671267e40c689e0bc00ca/conda-cheatsheet.pdf>

**NPS module used in CAFÉ-RNN**

Please refer to the **difference.txt** for addressing changes made during the study.

NPS main – contains

* setup file (setup.py)
* scripts
  + plot-training.sh -- plot loss/epoch for given channel using -c on log file
  + plot-hist.py -- feature histogram from pd.npy/gt.npy (prediction/ground truth)
  + animate-3d.py – feature animation from pd.npy/gt.npy
  + preprocess\_cafe.py – pre-process commands with example available here.
* NPS
  + main.py (**no changes done here** but starting point where all the arguments and processes are initiated)
  + trainer.py – Contains method for training/evaluate **(no changes made here)**
  + data – data loading and clip formation defined here **(no changes done)**
  + loss – loss functions definitions and classes **(no changes done here)**
  + model
    - ConvRNN
      * \_\_init\_\_().py – contains forward method and **predict\_next() [hard rules]**
      * **Predrnn\_v1.py – predictive RNN network**
* example job scripts (job script folder)

**Last revised: 08/08/2022**