Description of *phipredictor*

I. Installation

All code is encapsulated in a Python package called `phipredictor` to install it simply clone it or download it and run the command `python setup.py install`. However, if further development is intended then the use the command `python setup.py develop` instead, so that the package automatically updates when something changes in the source code. When developing it is also recommended the use of a virtual environment, consult the documentation.

II. Simulation

The simulation is handled by the class `PhaseSimulator` declared in the file `simulation.py`. The main interface with the class is made through the function `simulate` which takes 3 arguments:

1) *mirror_poses* - A matrix of size $(3, 4)$ which describes the poses of the 4 mirrors;
2) *noise* - A boolean, when true adds Poisson noise to the end result, otherwise has no effect;
3) *symmetry* - A boolean, when false removes a square from the south mirror to remove any symmetries, otherwise has no effect.

III. Random Generation

The generation of random samples is made using the class `RandomSampler` defined in the file `random_sampler.py`. This file can be directly run to generate datasets. The command `python random_sampler.py -h` shows all the options available, but the following is a brief overview.

- *output_dir* - Required. The resulting dataset is going to be saved;
- *-n N* - Number of examples to generate. Defaults to 500;
- *-p* - If passed poisson noise is addeed to the generated samples;
- *-s* - If passed symmetry is going to be eliminated in the generation.

The result of the generation consists a `csv` file where each line contains a file name and the corresponding mirror poses, and a folder containing the files matching the file names in the csv, each of these contains the simulation results.

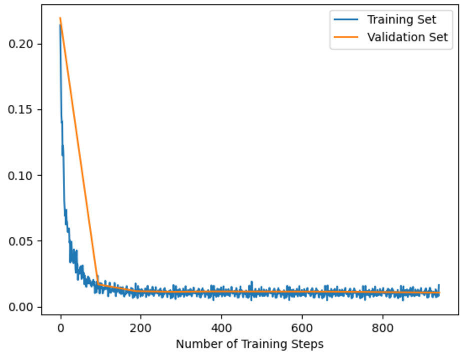

Figure 1: Training results with default values for learning rate and batch size while using a dataset without the removal of symmetry

IV. Model

The model to be fitted to the simulated data is implemented in the file `model.py` using `pytorch`.

V. Training

The training of the model can be done by executing the file `train.py`, this script has the following command line arguments:

1) *dataset* - Path to the root folder of the dataset;
2) *prefix* - Prefix of the folder name to which the results and metrics are going to be saved to;
3) *-lr LR* - Learning rate of the optimizer, defaults to 10^{-5};
4) *-epochs EPOCHS* - Number of epochs to train for, defaults to 10;
5) *-batch BATCH* - Batch size to use at each training step, defaults to 4;

As it stands the training as the following qualities:

- 4 fold cross-validation is used;
- The loss function corresponds to the Mean Square Error;
- At the end of each epoch the average loss is calculated for the entire validation fold;
- The optimizer corresponds to Stochastic Gradient Descent (provided by `pytorch`);
- The log function is applied to every example before it used for training, this is necessary due to the high magnitude of the values in the samples.

VI. Results

As it can be clearly seen in Figure **??** the model already converged after the first epoch.

VII. Problems