POLAR BACKGROUND PREDICTION

DOCUMENTATION: INDUSTRIAL REPORT

Stéphane Liem NGUYEN

Nicolas PRODUIT

Computer Science Department

DPNC: Nuclear and Corpuscular Physics Department

University of Geneva

October 4, 2023

Contents

1 Introduction **3**

2 Project structure **3**

3 Installation **4**

4 Usage **4**

4.1 Configuration file 5

4.2 Produced artifacts 10

5 Credits & useful links **11**

1 Introduction

Physicists from the POLAR collaboration are interested in detecting Gamma Ray Bursts (GRBs).

Based on past data collected from the POLAR detector attached to the Tiangong-2 spacelab, this project aims at building a model of the background and using it to extract potentially meaningful time intervals for further analysis by experts. These time intervals are extracted based on the magnitude of the difference between the target and predicted photon rates (photon counts per second) for different energy bins.

These time intervals might include moments in which the detector was turned off or had problems which caused our predictions to be significantly higher or lower than the target rates. They might also include solar events such as solar flares.

We recommend the user to read the abstract, introduction and appendices of our academic report for complementary information. Additional information on our dataset, methodology, discussion, and results are also there.

In contrast, the current report is for practical purposes and covers the GitHub project's structure, how we can use this project, and some information on the tools we've used.

2 Project structure

The project is composed of at least these files/folders:

- `./checkpoints`: configuration files, PyTorch checkpoints (model weights, optimizer and so on) for different runs
- `./config`: configuration file(s)
- `./data`: datasets in ROOT-CERN, `.pkl` or `.csv` format.
  This directory also contains 55 known GRB trigger times in `GRBs.csv` and a part of our results in either ROOT-CERN or `.pkl` format.
- `./logbook`: Markdown file explaining some details of what we've done each week
- `./notebooks`: Jupyter notebook using our trained model(s).
  The `./notebooks/README.md` explains how to get to the old Jupyter Notebooks and Python scripts used at the beginning of the project for data exploration/visualization and for creating basic models to predict photon rates (using, for instance, Scikit-learn Multi-Layer Perceptron).
- `README.md`

- `requirements.txt`
- `./results`
- `./src`: Python scripts for processing the data, training our model, visualizing our predictions, and other useful tasks

The `README.md` quickly explains what is needed to get started (installation, training phase, visualization of results). However, since some details might be missing, this report should act as a complement.

For the sake of completeness, we also explain again the content of the `README.md`.

3 Installation

- Follow the installation instructions from PyTorch. In particular, we used:

```
pip3 install torch torchvision torchaudio\
--index-url https://download.pytorch.org/whl/cu118
```

- And install other dependencies from `requirements.txt`:

```
pip3 install -r requirements.txt
```

- Download the `nf1rate.root` dataset in ROOT-CERN format from my Dropbox and place it under the `./data` directory.

We developed using Python `3.10.12` and `3.10.6` with torch `2.0.1` and torchvision `0.15.2`. Our code will not work on Windows, and using a different Python version might cause problems. As for the ROOT-CERN tool, we used ROOT `6.26/06`.

We also used Jupyter `v2023.7.1002162226` and Remote-SSH `v0.102.0` extensions of VSCode `1.81.1` to remotely edit and run codes on raidpolar (POLAR research group's Linux machine).

4 Usage

1. Place your ROOT-CERN file into the `./data` folder.
2. Change the `./config/trainer.yaml` config file with the correct filename under dataset.filename

- `python src/main.py` to run the training phase
- `python src/main.py wandb.mode=disabled` to run the training phase without using weights and biases

- `python src/visualizer.py` to load pre-trained model, plot loss and predicted photon rates (for the validation set if not specified).
- `python src/export.py` to export into .root format (ROOT CERN), our predictions over the whole dataset with the 25 known GRBs.
- `python src/export_cluster_intersections.py` to export, into .root format, the same as `python src/export.py` but also the cluster intersections for different sets of energy bins or conditions. Moreover, it also exports in .pkl different tables used in our academic report (incl. number of clusters).
- Run the different cells of `./notebooks/results.ipynb` to show the rest of our (interactive) plots (clusters, cluster intersections, etc.).

You can change the `./config/trainer.yaml` if you want a different model architecture, hyperparameters, features etc.

4.1 Configuration file

The following example[1] of `./config/trainer.yaml` configuration file is nearly self-explanatory about how to specify a different model architecture, hyperparameters, features etc.:

```yaml
verbose: False
wandb:
  project: POLAR-background-prediction
  mode: online
wandb_watch: True  # log in w&b model & criterion
common:
  seed: 42
  n_epochs: 200
  loss:
    name: null  # by default nn.MSELoss
  device: cuda
dataset:
  filename: data/nf1rate.root
  save_format: null  # won't save in .pkl nor .csv
  # new_columns: []
  new_columns:  # not necessarily used in the features or targets
```

[1] not necessarily the one we use

```
  - 1/rate_err[0]**2
  - rate[0]/rate_err[0]
feature_names:
  - unix_time
  - glat
  - glon
  - altitude
  - temperature
  - fe_cosmic
  - raz
  - decz
  - rax
  - decx
  - is_orbit_up
  - time_since_saa
  - crabarf
  - sun
  - sun_spot
  - B_r
  - B_theta
  - B_phi
target_names:
  - rate[0]/rate_err[0]  # had to specify in new_columns
filter_conditions:
  - rate[0]/rate_err[0] > 20
split:
  type: periodical
  periodicity: 200  # # of samples for the periodicity, for type periodical
train:
  size: 0.6
  batch_size: 200
  shuffle: True
val:
  size: 0.2
  batch_size: 200
test:
```

```
    size: 0.2
    batch_size: 200

model:
  type: MLP
  inner_activation_fct: ReLU
  output_activation_fct: null  # identity
  hidden_layer_sizes:
    - 100
    - 100

optimizer:
  hyperparams:
    # For adam optimizer
    lr: 1e-3
    betas:
      - 0.9
      - 0.999
    eps: 1e-08
    weight_decay: 0  # L2 regularization if > 0.
```

However, we still provide below some textual explanation for what the different parts mean:

`verbose`: boolean specifying whether we want to print additional information in the standard output.

`common`:

- `seed`: seed for random number generator used by PyTorch, random, and numpy.
- `n_epochs`: number of epochs (iterations through the whole training set). Although, it's the only end condition for the moment, an interesting end condition to add would be a stagnation end condition.
- `loss.name`: Possible choices of the loss function to use:
  - `null`: `nn.MSELoss` by default. Refer to our Appendix in the academic report and to the PyTorch documentation.
  - `weighted_mse_loss`: similar to MSELoss but instead of weighting each residual similarly, they can weight differently. Refer to the background model

subsection situated at the beginning of the methodology section in the academic report.

- `loss.weights`: specifies the weights $w_{i,j}$ for `weighted_mse_loss`. It can use columns created by `dataset.new_columns`.
- `device`: device on which PyTorch tensors are and on which operations on them are performed.

`dataset`:

- `filename`: path to our dataset, it can be in ROOT-CERN format or another supported format (.pkl or .csv). The behavior changes depending on the format. It's only in root format that our script uses `new_columns` and `filter_conditions`[2] but the other formats will load directly (e.g. no filtering) the data as Pandas DataFrame and work with it under the hood.
- `save_format`: saves or not the Pandas DataFrame containing the dataset in some format. It is only used when the filename is in ROOT-CERN format.

  Possible choices of `save_format`:

  - `null`: if don't want to save processed dataset
  - `pkl`: saves processed dataset in Pickle format (not split into train, validation and test set yet.)
  - `csv`: saves processed dataset in CSV format (not split into train, validation and test set yet.)

  TODO: We didn't try if it actually works with the CSV format
- `new_columns`: creates new columns in the Pandas DataFrame based on existing column names. These new columns can be used for filtering out examples or can be used as targets or features for our model. Order can matter.
- `feature_names`: name of the input features that are used in our model.
- `target_names`: name of the target(s)/output(s) that are used in our model (e.g. photon rates).
- `filter_conditions`: filter based on existing columns. It can be used to filter out known GRBs based on their `unix_time`. Order can matter.
- `split`: how to split the dataset into subsets such as train, validation, and test set.

  Possible choices of the `split.type`:

[2] In other words, it's only in root format that it goes through the processing steps of creating new columns and filtering out rows/examples

  - `random_split`[3]: uses PyTorch `random_split` to obtain a training, validation and test set (and PyTorch DataLoaders).
  - `periodical`: The dataset is periodically partitioned into train/val/test. In particular, each non-overlapping block of `split.periodicity` examples is contiguously split, without shuffling, into train/val/test subsets with the ratios specified in `[train|val|test].size`.

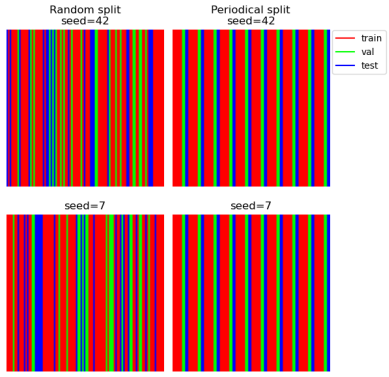

Figure 1: Random split vs. periodical split using $60/20/20\%$ split ratios and two initial seeds on a toy dataset with 100 examples.

- `[train|val|test].size`: The proportion of the data that goes into the training set, validation set and test set are specified by their `size` argument.
- `[train|val|test].batch_size`: mini-batch size used in the training, validation and test PyTorch DataLoaders. For instance, in the training phase, the model parameters are updated using `train.batch_size` examples instead of the whole training set. Note, however, that we still go through the entire training set for each epoch. Please refer to the section on Mini-batch Stochastic Gradient Descent in the Machine Learning Appendix from our academic report.
- Recall that the split is not saved in `save_format` format so every time we initialize our model (see `./src/trainer.py`, in the `init_datasets` method), it splits the dataset based on the what we wrote in the YAML configuration file.

[3]Actually, you can write anything and it will be `random_split` by default

`model`:

- `type`: choose which model to use

  Possible choice(s):

  - `MLP`: Multilayer perceptron (fully connected feedforward artificial neural network)

  Depending on the `model.type`, we can have different hyperparameters specifying the architecture.

  For instance, for `MLP`:

- `inner_activation_fct`: activation function used for the hidden layers.

  Possible choice(s):

  - `ReLU`: Rectified Linear Unit $\max(0, x) = x^+$

- `output_activation_fct`: activation function for the last output layer (by default identity function)
- `hidden_layer_sizes`: number of neurons for each hidden layer. You can use any number of hidden layers (at least one hidden layer) by adding more rows.

`optimizer.hyperparams`: Adam optimizer's hyper-parameters. See PyTorch Adam's documentation for what can be changed.

4.2 Produced artifacts

talk about exported prdictions, root files, how to use them, how to use pandas dataframes

5 Credits & useful links

- Code and project structure: https://github.com/eloialonso/iris
- 55 GRBs: https://www.researchgate.net/publication/326811280_Overview_of_the_GRB_observation_by_POLAR
- Code and project structure: https://github.com/eloialonso/iris
- LSTM to predict GRB occurrences from past photon counts and energy info: https://cernbox.cern.ch/pdf-viewer/public/X1OvZ4iY19vewcV/HAGRID_in_Space.pdf
- 55 GRBs: https://www.researchgate.net/publication/326811280_Overview_of_the_GRB_observation_by_POLAR
- POLAR collaboration
  - https://www.astro.unige.ch/polar/
  - https://www.unige.ch/dpnc/fr/groups/xin-wu/experiences/polar/
  - https://www.astro.unige.ch/polar/grb-light-curves
- Technologies
  - Weights & biases: https://wandb.ai/site
  - Pytorch:
    - https://pytorch.org/tutorials/beginner/basics/intro.html
    - https://pytorch.org/tutorials/recipes/recipes/tuning_guide.html
  - Hydra: https://hydra.cc/docs/intro/
  - ReviewNB ("git diff but for Jupyter notebooks"): https://www.reviewnb.com/
  - VSCode Remote SSH: https://code.visualstudio.com/docs/remote/ssh-tutorial
  - Diff with colors by using `diff <file1> <file2> --color`: https://man7.org/linux/man-pages/man1/diff.1.html
  - TMux detach/reattach session: https://www.redhat.com/sysadmin/introduction-tmux-linux