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

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/rate_err[0]**2
    - rate[0]/rate_err[0]
  feature_names:
    - unix_time
    - glat
    - glon
    - altitude
    - temperature
    - fe_cosmic
    - raz
    - decz
```

[1] not necessarily the one we use

```yaml
    - 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

[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

  - `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.)

- `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`:
  - `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`.

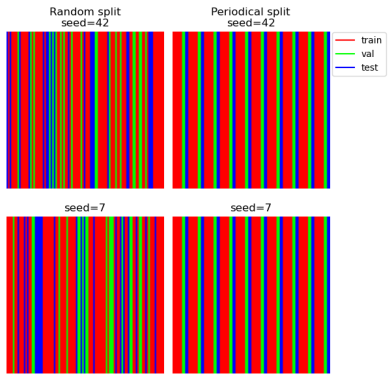

Figure 1: Random split vs. periodical split using $60/20/20\%$ split ratios and two initial seeds on a toy dataset with 100 examples.

- `[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.

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

- `[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.

`model`:

- `type`: choose which model to use

  Possible choice(s):

  - `MLP`: Multi-Layer 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 units 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

The following produce artifacts under the `./data` directory:

- `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.).

We will only focus on the second one as it produces the artifacts the others do.

4.2.1 ROOT-CERN files

We have produced ROOT-CERN files with these prefixes

- `pred_nf1rate`
- `cluster_inter_nf1rate`

where each unique suffix corresponds to a specific trained model[4].

We demonstrate below how to open `pred_nf1rate.root` and plot our predictions, shown in Figure 2, using the ROOT-CERN tool.

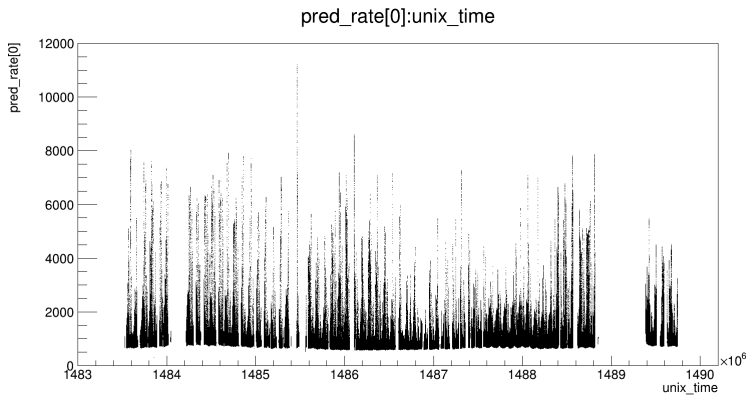

Figure 2: Predicted photon rate for the first energy bin

[4]When there's no suffix, it corresponds to the model trained with seed 42

```
zenchiyu@raidpolar3:~/POLAR-background-prediction (develop)$ ls
checkpoints  config  data  docs  documentation  LICENSE  logbook  notebooks  outputs  README.md  requirements.txt  results  scripts  src  wandb
zenchiyu@raidpolar3:~/POLAR-background-prediction (develop)$ cd data/
zenchiyu@raidpolar3:~/POLAR-background-prediction/data (develop)$ ls | grep pred_nf1rate
pred_nf1rate_0307eblk.root
pred_nf1rate_162pncmd.root
pred_nf1rate_45dljtwk.root
pred_nf1rate_5fs1oqli.root
pred_nf1rate_6wsxi1hf.root
pred_nf1rate_7sdm9qvf.root
pred_nf1rate_h444g60r.root
pred_nf1rate_ntlx23x5.root
pred_nf1rate.root
pred_nf1rate_tziecklt.root
pred_nf1rate_xetzm1dg.root
zenchiyu@raidpolar3:~/POLAR-background-prediction/data (develop)$ root pred_nf1rate.root
   ------------------------------------------------------------------
  | Welcome to ROOT 6.26/06                        https://root.cern |
  | (c) 1995-2021, The ROOT Team; conception: R. Brun, F. Rademakers |
  | Built for linuxx8664gcc on Jul 28 2022, 18:08:51                 |
  | From tags/v6-26-06@v6-26-06                                      |
  | With c++ (Ubuntu 11.2.0-19ubuntu1) 11.2.0                        |
  | Try '.help', '.demo', '.license', '.credits', '.quit'/'.q'       |
   ------------------------------------------------------------------

root [0]
Attaching file pred_nf1rate.root as _file0...
(TFile *) 0x564ea6acc080
root [1] .ls
TFile**         pred_nf1rate.root
 TFile*         pred_nf1rate.root
  KEY: TTree    pred_nf1rate;1
root [2] pred_nf1rate -> Print()
******************************************************************************
*Tree    :pred_nf1rate:                                                      *
*Entries :  3954837 : Total =       126556333 bytes  File  Size =   63980532 *
*        :          : Tree compression factor =   1.98                       *
******************************************************************************
*Br    0 :unix_time : unix_time/D                                            *
*Entries :  3954837 : Total  Size=   31639289 bytes  File Size  =   18836656 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=   1.68     *
*.............................................................................*
*Br    1 :pred_rate : pred_rate[6]/F                                         *
*Entries :  3954837 : Total  Size=   94916679 bytes  File Size  =   45142487 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=   2.10     *
*.............................................................................*
root [3] pred_nf1rate -> Draw("pred_rate[0]:unix_time")
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
root [4] Info in <TCanvas::Print>: pdf file /RAID/home/zenchiyu/POLAR-background-prediction/data/test.pdf has been created
Info in <TCanvas::Print>: pdf file /RAID/home/zenchiyu/pred_rate_0.pdf has been created
```

We can also display the start and end times of a particular positive cluster intersection:

```
zenchiyu@raidpolar3:~/POLAR-background-prediction/data (develop)$ root cluster_inter_nf1rate.root
   ------------------------------------------------------------------
  | Welcome to ROOT 6.26/06                        https://root.cern |
  | (c) 1995-2021, The ROOT Team; conception: R. Brun, F. Rademakers |
  | Built for linuxx8664gcc on Jul 28 2022, 18:08:51                 |
  | From tags/v6-26-06@v6-26-06                                      |
  | With c++ (Ubuntu 11.2.0-19ubuntu1) 11.2.0                        |
  | Try '.help', '.demo', '.license', '.credits', '.quit'/'.q'       |
   ------------------------------------------------------------------
root [0]
Attaching file cluster_inter_nf1rate.root as _file0...
(TFile *) 0x563d70406f40
root [1] .ls
TFile**         cluster_inter_nf1rate.root
 TFile*         cluster_inter_nf1rate.root
  KEY: TTree    cluster_inter_nf1rate;1
  KEY: TTree    negative_cluster_inter_nf1rate;1
root [2] cluster_inter_nf1rate -> Print()
******************************************************************************
*Tree    :cluster_inter_nf1rate:                                             *
*Entries :        2 : Total =          508743 bytes  File  Size =     309508 *
*        :          : Tree compression factor =   1.70                       *
******************************************************************************
*Br    0 :2_3       : 2_3[2815]/D                                            *
*Entries :        2 : Total  Size=      45624 bytes  File Size  =      20683 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=   2.18     *
*.............................................................................*
*Br    1 :2_3_4_5   : 2_3_4_5[3474]/D                                        *
*Entries :        2 : Total  Size=      56188 bytes  File Size  =      30122 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=   1.85     *
*.............................................................................*
# etc.
*.............................................................................*
*Br   40 :#_inter_more_3 : #_inter_more_3[3178]/D                            *
*Entries :        2 : Total  Size=      51487 bytes  File Size  =      29915 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=   1.70     *
*.............................................................................*
*Br   41 :#_inter_more_4 : #_inter_more_4[1422]/D                            *
*Entries :        2 : Total  Size=      23391 bytes  File Size  =      13224 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=   1.73     *
*.............................................................................*
*Br   42 :#_inter_more_5 : #_inter_more_5[1028]/D                            *
*Entries :        2 : Total  Size=      17087 bytes  File Size  =       9714 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=   1.70     *
*.............................................................................*
root [3] cluster_inter_nf1rate -> Scan("#_inter_more_3[0]", "", "colsize=19")
***********************************
*    Row   *   #_inter_more_3[0] *  # picked cluster n°0 out of 3178 clusters
***********************************
*        0 *  1483569641.4953258 *  # begin unix time of cluster 0
*        1 * 1483569644.49604344 *  # end unix time of cluster 0
***********************************
(long long) 2
```

Or a particular negative cluster intersection:

```
root [5] negative_cluster_inter_nf1rate -> Print()
******************************************************************************
*Tree    :negative_cluster_inter_nf1rate:                                      *
*Entries :        2 : Total =          563394 bytes  File  Size =     327226 *
*        :          : Tree compression factor =   1.78                       *
******************************************************************************
*Br    0 :2_3_4_5   : 2_3_4_5[3703]/D                                         *
*Entries :        2 : Total  Size=      59861 bytes  File Size  =      29422 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=   2.02     *
*.............................................................................*
*Br    1 :0_2_3_4_5 : 0_2_3_4_5[263]/D                                        *
*Entries :        2 : Total  Size=       4829 bytes  File Size  =       1892 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=   2.28     *
*.............................................................................*
# etc.
*.............................................................................*
*Br   39 :#_inter_more_3 : #_inter_more_3[3744]/D                            *
*Entries :        2 : Total  Size=      60552 bytes  File Size  =      30992 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=   1.94     *
*.............................................................................*
*Br   40 :#_inter_more_4 : #_inter_more_4[1261]/D                            *
*Entries :        2 : Total  Size=      20824 bytes  File Size  =      10370 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=   1.96     *
*.............................................................................*
*Br   41 :#_inter_more_5 : #_inter_more_5[833]/D                             *
*Entries :        2 : Total  Size=      13974 bytes  File Size  =       6901 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=   1.95     *
*.............................................................................*
root [6] negative_cluster_inter_nf1rate -> Scan("#_inter_more_3[0]", "", "colsize=19")
**********************************
*    Row   *   #_inter_more_3[0] *  # picked cluster n°0 out of 3744 clusters
**********************************
*        0 * 1483558662.53446245 *  # begin unix time of cluster 0
*        1 * 1483558662.53446245 *  # end unix time of cluster 0
**********************************
(long long) 2
root [7]
```

4.2.2 Pickled Pandas DataFrames

We also produced pickled Pandas DataFrames with these prefixes:

- `num_examples_clusters`
- `num_cluster_intersections`

that track the number of positive/negative examples or clusters for each energy bin and the number of cluster intersections for different sets of energy bins. Again, each unique suffix corresponds to a specific trained model.

We also demonstrate below how to read two of them from the Python interpreter:

```
zenchiyu@raidpolar3:~/POLAR-background-prediction/data (develop)$ python
Python 3.10.6 (main, May 29 2023, 11:10:38) [GCC 11.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import pandas as pd
>>> num_examples_clusters = pd.read_pickle("num_examples_clusters.pkl")
>>> num_examples_clusters
           new_std  # +ve examples  # -ve examples  # +ve clusters  # -ve clusters
rate[0]   1.344533            3952            2576            1507            1570
rate[1]   1.199649            5172            2834            1969            1682
rate[5]   1.322103           18408           15227            5500            6666
rate[6]   1.256458           20015           15480            5893            6807
rate[10]  1.506859           13121           10317            3967            4700
rate[12]  1.501043           13227           10423            4004            4745
>>> num_cluster_intersections = pd.read_pickle("num_cluster_intersections.pkl")
>>> num_cluster_intersections.head()
           negative  positive
2,3,4,5        3703      3474
0,2,3,4,5       263       150
1,2,3,4,5       426       718
2,3            2874      2815
3              1452      1573
>>> num_cluster_intersections.index
Index(['2,3,4,5', '0,2,3,4,5', '1,2,3,4,5', '2,3', '3', '0,1,2,3,4,5', '2,4,5',
       '2', '4,5', '2,3,4', '0,2,4,5', '0', '0,1,2,3', '1', '0,2', '1,2,3,5',
       '2,3,5', '5', '1,2,3', '0,2,3', '1,3', '0,4,5', '0,1,2', '2,5', '1,2',
       '3,4,5', '3,5', '4', '0,1,4,5', '0,1', '0,2,5', '1,2,4,5', '3,4',
       '1,4,5', '0,1,2,4,5', '2,4', '# inter > 0', '# inter > 1',
       '# inter > 2', '# inter > 3', '# inter > 4', '# inter > 5', '0,1,3,4,5',
       '1,3,4,5', '1,2,3,4', '0,1,2,3,4', '0,3,4,5'],
      dtype='object')
>>> num_cluster_intersections.loc["# inter > 3",:]
negative    3744
positive    3178
Name: # inter > 3, dtype: Int64
```

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