UNIVERSITY OF CENTRAL FLORIDA

MARGE Users Manual

MACHINE LEARNING ALGORITHM FOR RADIATIVE TRANSFER OF GENERATED EXOPLANETS

Authors:
Michael D. HIMES

Supervisor:
Dr. Joseph HARRINGTON

July 12, 2020

Contents

1 Team Members

- Michael Himes[1], University of Central Florida (mhimes@knights.ucf.edu)
- Joseph Harrington, University of Central Florida
- Adam Cobb, University of Oxford
- David C. Wright, University of Central Florida
- Zacchaeus Scheffer, University of Central Florida

2 Introduction

This document describes MARGE, the Machine learning Algorithm for Radiative transfer of Generated Exoplanets. MARGE generates exoplanet spectra; processes them into a desired format; and trains, validates, and tests neural network (NN) models to approximate radiative transfer (RT). At present, MARGE is configured to use BART, the Bayesian Atmospheric Radiative Transfer code, for spectra generation.

The detailed MARGE code documentation and User Manual[2] are provided with the package to assist users in its usage. For additional support, contact the lead author (see Section 1).

MARGE is released under the Reproducible Research Software License. For details, see https://planets.ucf.edu/resources/reproducible-research/software-license/.

The MARGE package is organized as follows:

```
MARGE
├── doc
├── example
├── lib
│   └── datagen
│       └── BART
└── modules
    └── BART
```

3 Installation

3.1 System Requirements

MARGE was developed on a Unix/Linux machine using the following versions of packages:

- Python 3.7.2
- Keras 2.2.4
- Numpy 1.16.2

[1] https://github.com/mdhimes/

[2] Most recent version of the manual available at https://exosports.github.io/MARGE/doc/MARGE_User_Manual.html

- Matplotlib 3.0.2
- mpi4py 3.0.3
- Scipy 1.2.1
- sklearn 0.20.2
- Tensorflow 1.13.1
- CUDA 9.1.85
- cuDNN 7.5.00
- ONNX 1.6.0
- keras2onnx 1.6.1
- onnx2keras 0.0.18

MARGE also requires a working MPI distribution if using BART for data generation. MARGE was developed using MPICH version 3.3.2.

3.2 Install and Compile

3.2.1 Unix

This is our recommended installation method. The following instructions have been verified on Ubuntu and may need to be slightly modified for other Unix distributions (e.g., Mac).
To begin, obtain the latest stable version of MARGE. Decide on a local directory to hold MARGE. Let the path to this directory be MARGE. Now, clone the repository:

```
git clone --recursive https://github.com/exosports/MARGE MARGE/
cd MARGE/
```

MARGE contains a file to easily build a conda environment capable of executing the software. Create the environment via

```
conda env create -f environment.yml
```

Then, activate the environment:

```
conda activate marge_homer
```

Users that do not have Nvidia GPU drivers installed will need to remove the tensorflow-gpu package:

```
conda remove -n marge_homer tensorflow-gpu
```

Follow the prompt to upgrade/downgrade relevant packages.
You are now ready to run MARGE!
Data generation with BART requires SWIG. For users that do not already have it installed, it can be easily installed via conda:

```
conda install swig
```

To use BART, its submodules must be compiled:

```
make bart
```

You are now ready to run MARGE with BART!

3.2.2 Windows

Windows is not yet fully supported. We advise that Windows 10 users follow the Unix instructions using the Windows Subsystem for Linux, which provides a Bash terminal.
The BART compilation scripts require a C compiler (e.g., gcc). Additionally, some of the compilation scripts are intended to use Bash. Problems can arise due to the return character used in Windows, so users may need to make use of a utility that converts line endings.
If a user finds a method to run MARGE in Windows, we encourage them to share their installation method to improve the documentation.

4 Example

The following script will walk a user through executing all modes of MARGE to simulate the emission spectra of an HD 189733 b-like exoplanet with a variety of thermal profiles and atmospheric compositions, process the data, and train an NN model to quickly approximate spectra over the trained parameter space. These instructions are meant to be executed from a Linux terminal.

We offer a lightweight example meant to be executed on a machine with 4 cores and 6 GB of RAM. Note that we are compromising the completeness and accuracy of the resulting model to ensure that the software can be executed in a reasonable amount of time (though it will still take a while!). To improve the results, users may increase the number of spectra generated (numit in BART config), use a finer opacity grid (reduce tempdelt and wndelt in BART config), increase the number of atmospheric layers (n_layers in BART config), and/or use a more complicated model architecture (more layers, more nodes).
Requirements:

- ¿= 4 cores
- ¿= 4 GB RAM available (recommended system RAM ¿= 6 GB)
- ¿= 20 GB free space

Optional:

- GPU with ¿= 4 GB RAM

To begin, copy the requisite files to another directory. Here we assume that directory is parallel to MARGE, called run. From the MARGE directory,

```
mkdir ../run
cp -a ./example/* ../run/.
cd ../run
```

To generate data with BART, a Transit Line-Information (TLI) file containing all line list information must be created. Download the required line lists and create the TLI file (this may take a while!):

```
./get_lists.sh
../MARGE/modules/BART/modules/transit/pylineread/src/pylineread.py -c pyline
```

Now, execute MARGE:

```
../MARGE/MARGE.py MARGE.cfg
```

This will take some time to run. It will

- generate an opacity table,
- run BART to generate spectra,
- process the generated spectra into MARGE's desired format,
- train, validate, and test an NN model, and
- plot specific predicted vs. true spectra.

Users can disable some steps via boolean flags within the configuration file. For details, see the following section.

5 Program Inputs

The executable MARGE.py is the driver for the MARGE program. It takes a a configuration file of parameters. Once configured, MARGE is executed via the terminal as described in Section 4.

5.1 MARGE Configuration File

The MARGE configuration file is the main file that sets the arguments for a MARGE run. The arguments follow the format **`argument = value`**, where **`argument`** is any of the possible arguments described below. Note that, if generating data via BART, the user must create a BART configuration file (see Section 5.1.1).
The available options for a MARGE configuration file are listed below.
(Directories)

- inputdir : str. Directory containing MARGE inputs.
- outputdir : str. Directory containing MARGE outputs.
- plotdir : str. Directory to save plots. If relative path, subdirectory within 'outputdir'.
- datadir : str. Directory to store generated data. If relative path, subdirectory within 'outputdir'.
- preddir : str. Directory to store validation and test set predictions and true values. If relative path, subdirectory within 'outputdir'.

Datagen Parameters

- datagen : bool. Determines whether to generate data.
- datagenfile: str. File containing functions to generate and/or process data. Do NOT include the .py extension! See the files in the lib/datagen directory for examples. User-specified files must have identically-named functions with an identical set of inputs. If an additional input is required, the user must modify the code in MARGE.py accordingly. Please submit a pull request if this occurs!

- cfile : str. Name of the configuration file for data generation. Can be absolute path, or relative path to ‘inputdir‘.
- processdat : bool. Determines whether to process the generated data.
- preservedat: bool. Determines whether to preserve the unprocessed data after completing data processing. Note: if False, it will PERMANENTLY DELETE the original, unprocessed data!

Neural Network (NN) Parameters

- nnmodel : bool. Determines whether to use an NN model.
- resume : bool. Determines whether to resume training a previously-trained model.
- seed : int. Random seed.
- trainflag : bool. Determines whether to train an NN model.
- validflag : bool. Determines whether to validate an NN model.
- testflag : bool. Determines whether to test an NN model.
- TFR_file : str. Prefix for the TFRecords files to be created.
- buffer : int. Number of batches to pre-load into memory.
- ncores : int. Number of CPU cores to use to load the data in parallel.
- normalize : bool. Determines whether to normalize the data by its mean and standard deviation.
- scale : bool. Determines whether to scale the data to be within a range.
- scalelims : floats. The min, max of the range of the scaled data. It is recommended to use -1, 1
- weight_file: str. File containing NN model weights. NOTE: MUST end in .h5
- input_dim : int. Dimensionality of the input to the NN.
- output_dim : int. Dimensionality of the output of the NN.
- ilog : bool. Determines whether to take the log10 of the input data. Alternatively, specify comma-, space-, or newline-separated integers to selectively take the log of certain inputs.
- olog : bool. Determines whether to take the log10 of the output data. Alternatively, specify comma-, space-, or newline-separated integers to selectively take the log of certain outputs.
- gridsearch : bool. Determines whether to perform a gridsearch over architectures.
- architectures: strings. Name/identifier for a given model architecture. Names must not include spaces! For multiple architectures, separate with a space or indented newlines.
- nodes : ints. Number of nodes per layer with nodes.

- layers: strings. Type of each hidden layer. Options: dense, conv1d, maxpool1d, avgpool1d, flatten, dropout
- lay_params: list. Parameters for each layer (e.g., kernel size). For no parameter or the default, use None.
- activations: strings. Type of activation function per layer with nodes. Options: linear, relu, leakyrelu, elu, tanh, sigmoid, exponential, softsign, softplus, softmax
- act_params: list. Parameters for each activation. Use None for no parameter or the default value. Values specified only apply to LeakyReLU and ELU.
- epochs : int. Maximum number of iterations through the training data set.
- patience : int. Early-stopping criteria; stops training after 'patience' epochs of no improvement in validation loss.
- batch_size : int. Mini-batch size for training/validation steps.
- lengthscale: float. Minimum learning rate.
- max_lr : float. Maximum learning rate.
- clr_mode : str. Specifies the function to use for a cyclical learning rate (CLR). 'triangular' linearly increases from 'lengthscale' to 'max_lr' over 'clr_steps' iterations, then decreases. 'triangular2' performs similar to 'triangular', except that the 'max_lr' value is decreased by 2 every complete cycle, i.e., 2 * 'clr_steps'. 'exp_range' performs similar to 'triangular2', except that the amplitude decreases according to an exponential based on the epoch number, rather than the CLR cycle.
- clr_steps : int. Number of steps through a half-cycle of the learning rate. E.g., if using clr_mode = 'triangular' and clr_steps = 4, Every 8 epochs will have the same learning rate. It is recommended to use an even value. For more details, see Smith (2015), Cyclical Learning Rates for Training Neural Networks.

Plotting Parameters

- xvals : str. .NPY file with the x-axis values corresponding to the NN output.
- xlabel : str. X-axis label for plots.
- ylabel : str. Y-axis label for plots.
- plot_cases : ints. Specifies which cases in the test set should be plotted vs. the true spectrum. Note: must be separated by spaces or indented newlines.

Statistics Files

- fmean : str. File name containing the mean of each input/output. Assumed to be in 'inputdir'.
- fstdev : str. File name containing the standard deviation of each input/output. Assumed to be in 'inputdir'.

- fmin : str. File name containing the minimum of each input/output. Assumed to be in ‘inputdir‘.
- fmax : str. File name containing the maximum of each input/output. Assumed to be in ‘inputdir‘.
- rmse_file : str. Prefix for the file to be saved containing the root mean squared error of predictions on the validation & test data. Saved into ‘outputdir‘.
- r2_file : str. Prefix for the file to be saved containing the coefficient of determination ($R\hat{2}$) of predictions on the validation & test data. Saved into ‘outputdir‘.
- filters : strings. (optional) Paths/to/filter files that define a bandpass over ‘xvals‘. Two columns, wavelength and transmission. Used to compute statistics for band-integrated values.
- filt2um : float. (default: 1.0) Factor to convert the filter files’ X values to microns. Only used if ‘filters‘ is specified.

5.1.1 BART Configuration File

The BART User Manual details the creation of a BART configuration file. For compatibility with MARGE, users must ensure two specific arguments are set within the configuration file:

- savemodel: base file name of the generated data. MUST have ’.npy’ file extension.
- modelper: an integer that sets the batch size of each ‘savemodel‘ file.

Note that ‘modelper‘ batch size corresponds to the iterations per chain. For example, if using 10 parallel chains, a ‘modelper‘ of 512 would save out files of 5120 spectra each.
Executing BART requires a Transit Line-Information (TLI) file to be created. For details on generating a TLI file, see the Transit User Manual. For an example, see Section 4.

6 Program Outputs

MARGE produces the following outputs if all modes are executed:

- simulated spectra.npy files
- processed spectra.npy files
- .NPY file of the mean of training set inputs and outputs
- .NPY file of the standard deviation of training set inputs and outputs
- .NPY file of the minima of training set inputs and outputs
- .NPY file of the maxima of training set inputs and outputs
- .NPY file of the size of the training, validation, and test sets
- TFRecords files of the data set

- file containing the NN model and weights
- predicted and true spectra
- RMSE and $R\hat{2}$ statistics
- plots of true and predicted spectra

Note that BART's output files are not discussed here; see the BART User Manual for details.

7 Be Kind

Please cite this paper if you found this package useful for your research:

- Himes et al. (2020), submitted to PSJ.

```
@article{HimesEtal2020psjMARGEHOMER,
   author = {{Himes}, Michael D. and {Harrington}, Joseph and {Cobb}, Adam D
         {O'Beirne}, Molly D. and {Zorzan}, Simone and
         {Wright}, David C. and {Scheffer}, Zacchaeus and
         {Domagal-Goldman}, Shawn D. and {Arney}, Giada N.},
    title = "Accurate Machine Learning Atmospheric Retrieval via a Neural Ne
  journal = {PSJ},
     year = 2020,
    pages = {submitted to PSJ}
}
```

Thanks!