pybaobab

Release 0.1

Jan 28, 2021

Contents

Training data generator for hierarchically modeling strong lenses with Bayesian neural networks

The `baobab` package can generate images of strongly-lensed systems, given some configurable prior distributions over the parameters of the lens and light profiles as well as configurable assumptions about the instrument and observation conditions. It supports prior distributions ranging from artificially simple to empirical.

A major use case for `baobab` is the generation of training and test sets for hierarchical inference using Bayesian neural networks (BNNs). The idea is that Baobab will generate the training and test sets using different priors. A BNN trained on the training dataset learns not only the parameters of individual lens systems but also, implicitly, the hyperparameters describing the training set population (the training prior). Such hierarchical inference is crucial in scenarios where the training and test priors are different, so that techniques such as importance weighting can be employed to bridge the gap in the BNN response.

CHAPTER 1

Installation

0. You'll need a Fortran compiler and Fortran-compiled *fastell4py*, which you can get on a debian system by running

```
$sudo apt-get install gfortran
$git clone https://github.com/sibirrer/fastell4py.git <desired location>
$cd <desired location>
$python setup.py install --user
```

1. Virtual environments are strongly recommended, to prevent dependencies with conflicting versions. Create a conda virtual environment and activate it:

```
$conda create -n baobab python=3.6 -y
$conda activate baobab
```

2. Now do one of the following.

Option 2(a): clone the repo (please do this if you'd like to contribute to the development).

```
$git clone https://github.com/jiwoncpark/baobab.git
$cd baobab
$pip install -e . -r requirements.txt
```

Option 2(b): pip install the release version (only recommended if you're a user).

```
$pip install baobab
```

3. (Optional) To run the notebooks, add the Jupyter kernel.

```
$python -m ipykernel install --user --name baobab --display-name "Python (baobab)"
```

4. (Optional) To enable online data augmentation for machine learning, install the relevant dependencies.

```
$pip install torch torchvision
$pip install tensorflow-gpu
```

CHAPTER 2

Usage

1. Choose your favorite config file among the templates in the *configs* directory and *copy* it to a directory of your choice, e.g.

```
$mkdir my_config_collection
$cp baobab/configs/tdlmc_diagonal_config.py my_config_collection/my_config.py
```

2. Customize it! You might want to change the *name* field first with something recognizable. Pay special attention to the *components* field, which determines which components of the lensed system (e.g. lens light, AGN light) become sampled from relevant priors and rendered in the image.
3. Generate the training set, e.g. continuing with the example in #1,

```
$generate my_config_collection/my_config.py
```

Although the *n_data* (size of training set) value is specified in the config file, you may choose to override it in the command line, as in

```
$generate my_config_collection/my_config.py 100
```

CHAPTER 3

Feedback

Please message @jiwoncpark with any questions.

There is an ongoing document that details our BNN prior choice, written and maintained by Ji Won.

CHAPTER 4

Attribution

`baobab` heavily uses `lenstronomy`, a multi-purpose package for modeling and simulating strongly-lensed systems (see source). When you use `baobab` for your project, please cite `lenstronomy` with Birrer & Amara 2018 as well as Park et al. 2019 (in prep).

Contents:

4.1 Installation

0. You'll need a Fortran compiler and Fortran-compiled *fastell4py*, which you can get on a debian system by running

```
$sudo apt-get install gfortran
$git clone https://github.com/sibirrer/fastell4py.git <desired location>
$cd <desired location>
$python setup.py install --user
```

1. Virtual environments are strongly recommended, to prevent dependencies with conflicting versions. Create a conda virtual environment and activate it:

```
$conda create -n baobab python=3.6 -y
$conda activate baobab
```

2. Now do one of the following.

Option 2(a): clone the repo (please do this if you'd like to contribute to the development).

```
$git clone https://github.com/jiwoncpark/baobab.git
$cd baobab
$pip install -e . -r requirements.txt
```

Option 2(b): pip install the release version (only recommended if you're a user).

```
$pip install pybaobab
```

3. (Optional) To run the notebooks, add the Jupyter kernel.

```
$python -m ipykernel install --user --name baobab --display-name "Python (baobab)"
```

4. (Optional) To enable online data augmentation for machine learning, install the relevant dependencies.

```
$pip install torch torchvision
$pip install tensorflow-gpu
```

4.2 Usage

1. Choose your favorite config file among the templates in the *configs* directory and *copy* it to a directory of your choice, e.g.

```
$mkdir my_config_collection
$cp baobab/configs/tdlmc_diagonal_config.py my_config_collection/my_config.py
```

2. Customize it! You might want to change the *name* field first with something recognizable. Pay special attention to the *components* field, which determines which components of the lensed system (e.g. lens light, AGN light) become sampled from relevant priors and rendered in the image.
3. Generate the training set, e.g. continuing with the example in #1,

```
$generate my_config_collection/my_config.py
```

Although the *n_data* (size of training set) value is specified in the config file, you may choose to override it in the command line, as in

```
$generate my_config_collection/my_config.py 100
```

4.3 baobab package

4.3.1 Subpackages

baobab.configs package

baobab.configs.parser module

class baobab.configs.parser.**BaobabConfig**(*user_cfg*)
Bases: object

Nested dictionary representing the configuration for Baobab data generation

export_log()
Export the baobab log to the current working directory

classmethod from_file(*user_cfg_path*)
Alternative constructor that accepts the path to the user-defined configuration python file :param user_cfg_path: path to the user-defined configuration python file :type user_cfg_path: str or os.path object

get_noise_kwargs(*bandpass*)
Return the noise kwargs defined in the babobab config, e.g. for passing to the noise model for online data augmentation

Returns

- **(dict)** (*A dict containing the noise kwargs to be passed to the noise*) – model.
- **(str)** (*The bandpass to pull the noise information for*)

get_survey_info(*survey_info*, *psf_type*)
Fetch the camera and instrument information corresponding to the survey string identifier

interpret_kinematics_cfg()
Validate the kinematics config

interpret_magnification_cfg()

baobab.bnn_priors package

baobab.bnn_priors.base_bnn_prior module

class baobab.bnn_priors.base_bnn_prior.**BaseBNNPrior**(*bnn_omega*, *components*)
Bases: abc.ABC

Abstract base class equipped with PDF evaluation and sampling utility functions for various lens/source macromodels

eval_param_pdf(*eval_at*, *hyperparams*)
Assigns and evaluates the PDF

sample()
Gets kwargs of sampled parameters to be passed to lenstronomy

Overridden by subclasses.

sample_param(*hyperparams*)
Assigns a sampling distribution

set_comps_qphi_to_e1e2()

set_params_list(*params_to_exclude*)
Set the list of tuples, each tuple specifying the component and parameter name, to be realized independently as well as the list of tuples to be converted from the q, phi convention to the e1, e2 convention

baobab.bnn_priors.diagonal_bnn_prior module

class baobab.bnn_priors.diagonal_bnn_prior.**DiagonalBNNPrior**(*bnn_omega*, *components*)
Bases: *baobab.bnn_priors.base_bnn_prior.BaseBNNPrior*

BNN prior with independent parameters

Note: This BNNPrior is cosmology-agnostic. For a version that's useful for H0 inference, see *DiagonalCosmoBNNPrior*.

---

sample()
Gets kwargs of sampled parameters to be passed to lenstronomy

Returns dictionary of config-specified components (e.g. lens mass), itself a dictionary of sampled parameters corresponding to the config-specified profile of that component

Return type dict

baobab.bnn_priors.cov_bnn_prior module

class baobab.bnn_priors.cov_bnn_prior.**CovBNNPrior**(*bnn_omega*, *components*)
Bases: *baobab.bnn_priors.base_bnn_prior.BaseBNNPrior*

BNN prior with marginally covariant parameters

Note: This BNNPrior is cosmology-agnostic. For a version that's useful for H0 inference, see *CovCosmoBNNPrior*.

---

sample()
Gets kwargs of sampled parameters to be passed to lenstronomy

Returns dictionary of config-specified components (e.g. lens mass), itself a dictionary of sampled parameters corresponding to the config-specified profile of that component

Return type dict

baobab.bnn_priors.empirical_bnn_prior module

class baobab.bnn_priors.empirical_bnn_prior.**EmpiricalBNNPrior**(*bnn_omega*, *components*)
Bases: *baobab.bnn_priors.base_bnn_prior.BaseBNNPrior*, baobab.bnn_priors.base_cosmo_bnn_prior.BaseCosmoBNNPrior

BNN prior that encodes physical correlations between parameters

get_agn_absolute_magnitude(*z_src*)
Get the AGN absolute magnitude at 1450A, sampled from the luminosity function for its redshift bin

Parameters **z_src** (*float*) – the AGN redshift

Returns AGN absolute magnitude at 1450A

Return type float

get_lens_absolute_magnitude(*vel_disp*)
Get the lens absolute magnitude from the Faber-Jackson relation given the realized velocity dispersion, with some scatter

Parameters **vel_disp** (*float*) – the velocity dispersion in km/s

Returns the V-band absolute magnitude

Return type float

get_lens_apparent_magnitude(*M_lens*, *z_lens*)
Get the lens apparent magnitude from the Faber-Jackson relation given the realized velocity dispersion, with some scatter

Parameters

- **M_lens** (*float*) – the V-band absolute magnitude of lens
- **z_lens** (*float*) – the lens redshift

Note: Does not account for peculiar velocity or dust. K-correction is approximate and implicit, as the absolute magnitude is in the V-band (480nm ~ 650nm) and, for z ~ 2-3, this portion of the SED roughly lands in the IR.

Returns the apparent magnitude in the IR

Return type float

get_lens_size(*vel_disp*, *z_lens*, *m_V*)
Get the lens V-band efefctive radius from the Fundamental Plane relation given the realized velocity dispersion and apparent magnitude, with some scatter

Parameters

- **vel_disp** (*float*) – the velocity dispersion in km/s
- **z_lens** (*float*) – redshift
- **m_V** (*float*) – V-band apparent magnitude

Returns the effective radius in kpc and arcsec

Return type tuple

get_src_absolute_magnitude(*z_src*)
Sample the UV absolute magnitude from the luminosity function for the given redshift and convert into apparent magnitude

Parameters **z_src** (*float*) – the source redshift

Returns the absolute magnitude at 1500A

Return type float

get_src_apparent_magnitude(*M_src*, *z_src*)
Convert the souce absolute magnitude into apparent magnitude

Parameters

- **M_src** (*float*) – the source absolute magnitude
- **z_src** (*float*) – the source redshift

Note: Does not account for peculiar velocity or dust. K-correction is approximate and implicit, as the absolute magnitude is at 150nm and, for z ~ 5-9, this portion of the SED roughly lands in the IR.

Returns the apparent magnitude in the IR

Return type float

get_src_size(*z_src*, *M_V_src*)
Get the effective radius of the source from its empirical relation with V-band absolute magnitude and redshift

Parameters

- **M_V_src** (*float*) – V-band absolute magnitude of the source
- **z_src** (*float*) – source redshift

Returns tuple of the effective radius in kpc and arcsec

Return type tuple

sample()
Gets kwargs of sampled parameters to be passed to lenstronomy

Returns dictionary of config-specified components (e.g. lens mass), itself a dictionary of sampled parameters corresponding to the config-specified profile of that component

Return type dict

sample_vel_disp(*vel_disp_cfg*)
Sample velocity dispersion from the config-specified model, on a grid with the range and resolution specified in the config

Parameters **vel_disp_cfg** (*dict*) – Copy of *cfg.bnn_omega.kinematics.vel_disp*

Returns a realization of velocity dispersion

Return type float

baobab.bnn_priors.kinematics_models module

baobab.bnn_priors.kinematics_models.**vel_disp_function_CPV2007**(*vel_disp_grid*)
Evaluate the velocity dispersion function from the fit on SDSS DR6 by [1]_ on a provided grid and normalizes the result to unity, so it can be used as a PMF from which to draw the velocity dispersion.

Parameters **vel_disp_grid** (*array-like*) – a grid of velocity dispersion values in km/s

---

Note: The returned array is normalized to unity and we treat it as a PMF from which to sample the velocity dispersion. We also use the exact fit values also used in LensPop ([2]_).

References

Returns the velocity dispersion function evaluated at *vel_disp_grid*

Return type array-like, same shape as *vel_disp_grid*

baobab.bnn_priors.parameter_models module

baobab.bnn_priors.parameter_models.**approximate_theta_E_for_SIS**(*vel_disp_iso*, *z_lens*, *z_src*, *cosmo*)
Compute the Einstein radius for a given isotropic velocity dispersion assuming a singular isothermal sphere (SIS) mass profile

Parameters

- **vel_disp_iso** (*float*) – isotropic velocity dispersion, or an approximation to it, in km/s
- **z_lens** (*float*) – the lens redshift
- **z_src** (*float*) – the source redshift
- **cosmo** (*astropy.cosmology object*) – the cosmology

Note: The computation is purely analytic.

Returns the Einstein radius for an SIS in arcsec

Return type float

class baobab.bnn_priors.parameter_models.**FaberJackson**(*slope=None*, *intercept=None*, *fit_data=None*)

Bases: object

Represents the Faber-Jackson (FJ) relation between velocity dispersion and luminosity of elliptical galaxies.

FJ is a projection of the Fundamental Plane (FP) relation.

get_luminosity(*vel_disp*)

Evaluate the V-band luminosity L_V expected from the FJ relation for a given velocity dispersion

Parameters **vel_disp** (*float*) – the velocity dispersion in km/s

Returns log(L_V/L_solar)

Return type float

class baobab.bnn_priors.parameter_models.**FundamentalPlane**(*a=None*, *b=None*, *c=None*, *intrinsic_scatter=0.0*, *fit_data=None*)

Bases: object

Represents the Fundamental Plane (FP) relation between the velocity dispersion, luminosity, and effective radius for elliptical galaxies

Luminosity is expressed as apparent magnitude in this form.

get_effective_radius(*vel_disp*, *m_V*)

Evaluate the size expected from the FP relation for a given velocity dispersion and V-band apparent magnitude

Parameters

- **vel_disp** (*float*) – the velocity dispersion in km/s
- **m_V** (*float*) – the apparent V-band magnitude

Returns the effective radius in kpc

Return type float

class baobab.bnn_priors.parameter_models.**FundamentalMassHyperplane**(*a=None*, *b=None*, *intrinsic_scatter=0.0*, *fit_data=None*)

Bases: object

Represents bivariate relations (projections) within the Fundamental Mass Hyperplane (FMHP) relation between the stellar mass, stellar mass density, effective radius, and velocity dispersion of massive ETGs.

Only the relation between the power-law mass slope (gamma) and effective radius is currently supported.

get_gamma_from_R_eff(*R_eff*)

Evaluate the power-law slope of the mass profile from its power-law relation with effective radius

Parameters **R_eff** (*float*) – the effective radius in kpc

Returns the power-law slope, gamma

Return type float

get_gamma_from_vel_disp(*vel_disp*)
Evaluate the power-law slope of the mass profile from its power-law relation with effective radius

Parameters **vel_disp** (*float*) – the velocity dispersion in km/s

Returns the power-law slope, gamma

Return type float

class baobab.bnn_priors.parameter_models.**AxisRatioRayleigh**(*a=None*, *b=None*, *lower=0.2*, *fit_data=None*)
Bases: object

Represents various scaling relations that the axis ratio can follow with quantities like velocity dispersion, when its PDF is assumed to be a Rayleigh distribution

Only the relation with velocity dispersion is currently supported.

get_axis_ratio(*vel_disp*)
Sample (one minus) the axis ratio of the lens galaxy from the Rayleigh distribution with scale that depends on velocity dispersion

Parameters **vel_disp** (*float*) – velocity dispersion in km/s

Returns the axis ratio q

Return type float

baobab.bnn_priors.parameter_models.**redshift_binned_luminosity_function**(*z*, *M_grid*)
Sample FUV absolute magnitude from the redshift-binned luminosity function

Parameters

- **z** (*float*) – galaxy redshift
- **M_grid** (*array-like*) – grid of FUV absolute magnitudes at which to evaluate luminosity function

---

Note: For z < 4, we use the Schechter function fits in Table 1 of [1]_ and, for 4 < z < 8, those in Table 4 of [2]_. z > 8 are binned into the z=8 bin. I might add high-redshift models, e.g. from [3]_.

References

Returns unnormalized function of the absolute magnitude at 1500A

Return type array-like

baobab.bnn_priors.parameter_models.**size_from_luminosity_and_redshift_relation**(*z*, *M_V*)
Sample the effective radius of Lyman break galaxies from the relation with luminosity and redshift

Parameters

- **z** (*float*) – galaxy redshift

- **M_V** (*float*) – V-band absolute magnitude

Note: The relation and scatter agree with [1]_ and [2]_, which both show that size decreases with higher redshift. They have been used in LensPop ([3]_) for source galaxies.

References

Returns a sampled effective radius in kpc

Return type float

class baobab.bnn_priors.parameter_models.**AGNLuminosityFunction**(*M_grid*, *z_bins=None*, *alphas=None*, *betas=None*, *M_stars=None*, *fit_data=None*)

Bases: object

Redshift-binned AGN luminosity function parameterized as a double power-law

get_double_power_law(*alpha*, *beta*, *M_star*)

Evaluate the double power law at the given grid of absolute magnitudes

Parameters

- **alpha** (*float*) – bright-end slope of the double power-law luminosity function
- **beta** (*float*) – faint-end slope of the double power-law luminosity function
- **M_star** (*float*) – break magnitude

Note: Returned luminosity function is normalized to unity. See Note under *slope of the double power-law luminosity function*.

Returns the luminosity function evaluated at *self.M_grid* and normalized to unity

Return type array-like

sample_agn_luminosity(*z*)

Sample the AGN luminosity from the redshift-binned luminosity function

Parameters **z** (*float*) – the AGN redshift

Returns sampled AGN luminosity at 1450A in mag

Return type float

baobab.generate script

Generating the training data.

This script generates the training data according to the config specifications.

Example

To run this script, pass in the desired config file as argument:

```
$ generate baobab/configs/tdlmc_diagonal_config.py --n_data 1000
```

baobab.generate.**main**()

baobab.generate.**parse_args**()
Parse command-line arguments

baobab.to_hdf5 script

Converting .npy image files and metadata into HDF5

This script converts the baobab data into the HDF5 format.

Example

To run this script, pass in the baobab out_dir path as the first argument and the framework format as the second, e.g.:

```
$ to_hdf5 out_data/tdlmc_train_EmpiricalBNNPrior_seed1113 --format 'tf'
```

The output file will be named *tdlmc_train_EmpiricalBNNPrior_seed1113.h5* and can be found inside the directory provided as the first argument.

See the demo notebook *demo/Read_hdf5_file.ipynb* for instructions on how to access the datasets in this file.

baobab.to_hdf5.**main**()

baobab.to_hdf5.**parse_args**()
Parses command-line arguments

4.4 Feedback

Suggestions are always welcome! If you encounter an issue or areas for improvement, please message @jiwoncpark or make an issue.

Python Module Index

b

Index