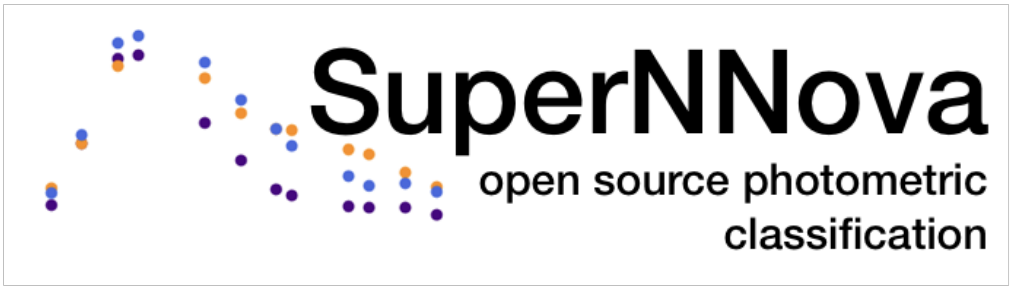
SuperNNova

unknown

Nov 18, 2021

GETTING STARTED


SuperNNova
open source photometric
classification

CHAPTER

ONE

SYSTEM CONFIGURATION

- This code has been tested on Ubuntu 16.04.
- For other configurations, support is not guaranteed.

The watermark of the test system is:

```
CPython 3.6.1
IPython 6.1.0

compiler   : GCC 4.4.7 20120313 (Red Hat 4.4.7-1)
system     : Linux
release    : 4.10.0-38-generic
machine    : x86_64
processor  : x86_64
CPU cores  : 8
interpreter: 64bit
CPU        : Intel(R) Xeon(R) CPU E5-1620 v3 @ 3.50GHz
GPU        : GeForce GTX 1080
```

CHAPTER

TWO

ENVIRONMENT CONFIGURATION

2.1 Conda virtual env

The preferred option to setup your environment is through conda environment as follows:

```
conda create --name <env> --file conda_env.txt
```

example configuration files for linux-64 (cpu and gpu) and osx-64 are provided in `SuperNNova/env`.

2.2 Docker

You can also use docker:

- Install docker: Docker.

Create a docker image:

```
cd env && make {image}
```

where `image` is one of `cpu` or `gpu` (for cuda 9.) or `gpu10` (for cuda 10.)

- This images contains all of this repository's dependencies.
- Image construction will typically take a few minutes

Enter docker environment by calling:

```
python launch_docker.py --image <image> --dump_dir </path/to/data>
```

- Add `--image image` where image is `cpu` or `gpu` (for cuda 9.) or `gpu10` (for cuda 10.)
- Add `--dump_dir /path/to/data` to mount the folder where you stored the data (see *Data walkthrough*) into the container. If unspecified, will use the default location (i.e. `snndump`)

This will launch an interactive session in the docker container, with zsh support.

CHAPTER

THREE

QUICKSTART GUIDE (GITHUB)

Welcome to SuperNNova!

This is a quick start guide so you can start testing our framework. If you want to install SuperNNova as a module, please take a look at *Quickstart guide (pip)*.

3.1 Installation

3.1.1 Clone the GitHub repository

```
git clone https://github.com/supernnova/supernnova.git
```

3.1.2 Setup your environment. 3 options

a) Create a docker image: *Docker* .

b) Create a conda virtual env *Environment configuration* .

c) Install packages manually. Inspect `conda_env.txt` for the list of packages we use.

3.2 Usage

For quick tests, a database that contains a limited number of light-curves is provided. It is located in `tests/raw`. For more information on the available data, check *Data walkthrough*.

3.2.1 Using command line

Build the database .. code:

```
python run.py --data --dump_dir tests/dump --raw_dir tests/raw --fits_dir tests/fits
```

Train an RNN

```
python run.py --train_rnn --dump_dir tests/dump
```

With this command you are training and validating our Baseline RNN with the test database. The trained model will be saved in a newly created model folder inside `tests/dump/models`.

The model folder has been named as follows: `vanilla_S_0_CLF_2_R_None_saltfit_DF_1.0_N_global_lstm_32x2_0.05_128_True_mean` (See below for the naming conventions). This folder's contents are:

- **saved model** (`*.pt`): PyTorch RNN model.
- **statistics** (`METRICS*.pickle`): pickled Pandas DataFrame with accuracy and other performance statistics for this model.
- **predictions** (`PRED*.pickle`): pickled Pandas DataFrame with the predicitons of our model on the test set.
- **figures** (`train_and_val_*.png`): figures showing the evolution of the chosen metric at each training step.

Remember that our data is split in training, validation and test sets.

You have trained, validated and tested your model. You can now inspect the test light-curves and their predictions in `tests/dump/lightcurves`.

3.2.2 Using Yaml

Build the database .. code:

```
python run_yml.py configs_yml/default.yml --mode data
```

Train an RNN

```
python run_yml.py configs_yml/default.yml --mode train_rnn
```

Available modes: `data`,``train_rnn``, `validate_rnn`, `plot_lcs`. Currently RF classification is not suppported with the yaml configurations. An example of classification using existing model is in `configs_yml/classify.yml`.

3.3 Reproduce SuperNNova paper

To reproduce the results of the paper please use the branch `paper` and run:

```
cd SuperNNova && python run_paper.py --debug --dump_dir tests/dump
```

`--debug` will train simplified models with a reduced number of epochs. Remove this flag for full reproducibility. With the `--debug` flag on, this should take between 15 and 30 minutes on the CPU.

3.4 Naming conventions

- **vanilla/variational/bayesian**: The type of RNN to be trained. `variational` and `bayesian` are bayesian recurrent networks
- **S_0**: seed used for training. Default is 0.
- **CLF_2**: number of targets to be used in classification. This case uses two classes: type Ia supernovae vs. all others.
- **R_None**: host-galaxy redshift provided. Options: `zpho` (photometric) or `zspe` (spectroscopic)

- **saltfit**: data used. In our database we split light-curves that have a succesful SALT2 fit (`saltfit`) and the complete dataset (`photometry`).
- **DF_1.0**: data fraction used in training. With large datasets it is usefult to test training with a fraction of the available training set. In this case we use the whole dataset (`1.0`).
- **N_global**: normalization used. Default: `global`.
- **lstm**: type of layer used. Default `lstm`.
- **32x2**: hidden layer dimension x number the layers.
- **0.05**: dropout value.
- **128**: batch size.
- **True**: if this model is bidirectional.
- **mean**: output option. `mean` is mean pooling.

The naming convention is defined in `SuperNNova/conf.py`.

CHAPTER

FOUR

QUICKSTART GUIDE (PIP)

Welcome to SuperNNova! This is a quick start guide so you can start testing our framework. This guide assumes you have installed it with pip, if you want to use the GitHub cloning please refer to *Quickstart guide (GitHub)*. Pip uses the master branch (not the frozen paper one and may be behind some updates).

4.1 Installation

4.1.1 Pip install

```
pip install supernnova
```

Please beware that SuperNNova only runs properly in Unix systems.

4.1.2 Setup your environment. 3 options

a) Create a conda virtual env *Environment configuration* (preferred).

b) Create a docker image: *Docker* .

c) Install packages manually. Inspect `conda_env.txt` for the list of packages we use.

4.2 Usage

For quick tests, a database that contains a limited number of light-curves is provided. It is located in `tests/raw`. For more information on the available data, check *Data walkthrough*. An example of running as module can be found in `sandbox/example_run_moduler_snn.py`.

4.2.1 Build the database

In the parent folder, where `run.py` is located you can launch python or ipython with the following:

```
import supernnova.conf as conf
from supernnova.data import make_dataset

# get config args
args =  conf.get_args()
```

(continues on next page)

(continued from previous page)

```
# create database
args.data = True                          # conf: making new dataset
args.dump_dir = "tests/dump"              # conf: where the dataset will be saved
args.raw_dir = "tests/raw"                # conf: where raw photometry files are saved
args.fits_dir = "tests/fits"              # conf: where salt2fits are saved
settings = conf.get_settings(args)        # conf: set settings
make_dataset.make_dataset(settings)       # make dataset
```

4.2.2 Train an RNN

```
import supernnova.conf as conf
from supernnova.training import train_rnn

# get config args
args =  conf.get_args()

args.train_rnn = True                     # conf: train rnn
args.dump_dir = "tests/dump"              # conf: where the dataset is saved
args.nb_epoch = 2                         # conf: training epochs
settings = conf.get_settings(args)        # conf: set settings
train_rnn.train(settings)                 # train rnn
```

4.2.3 Validate an RNN

```
import supernnova.conf as conf
from supernnova.validation import validate_rnn

# get config args
args =  conf.get_args()

args.validate_rnn = False                       # conf: validate rnn
args.dump_dir = "tests/dump"                    # conf: where the dataset is saved
settings = conf.get_settings(args)              # conf: set settings
validate_rnn.get_predictions(settings)          # classify test set
```

CHAPTER

FIVE

USING THE ON THE FLY CLASSIFICATION

After installing SuperNNova, you can use on the fly functionality.

The goal is to be able to input manually or load light-curve(s) from a file and classify them using a pre-trained model. This option does not build auxiliary files such as the database and is designed for small datasets.

5.1 Usage

Use the skeleton provided in `run_onthefly.py` to design your application.

CHAPTER

SIX

FAQ

6.1 General questions

- **What is SuperNNova?**

SuperNNova is a framework for lightcurve classification which uses supervised learning algorithms. Training of these algorithms rely on large annotated databases. Typically, we use simulations as the training set.

- **Do you have a paper describing SuperNNova? How can I cite you?**

The paper has been acccepted by MNRAS. A copy of the paper can be found here ArXiv.

- **How can I install it?**

You can either `clone` our GitHub or use `pip` for modular installation. Beware, the supported version of GitHub repository is this GitHub!!!! (previous version was hosted in a different webpage).

- **What data do I need?**

You only need lightcurves (photometric time-series) to use SuperNNova. Additional information can be added as well. E.g. we used supernova host-galaxy redshifts in the paper.

- **Is the data used in the paper publicly available?**

Yes it is! SuperNNovaSimulations We want to foster reproducibility so you can copy the data and reproduce all our experiments with `run_paper.py` in the `paper` branch. Beware, it will take while!

- **How did you create the simulations used in the paper?**

We used SNANA to generate the supernovae lightcurves. Our data is similar to the Supernova Photometric Classification Challenge (SPCC) data with updated models used in the DES simulations.

- **Why use SuperNNova?**

First, it is open source, so you can modify it for your science goal or just see for yourself what is the "blackbox". Second, we have pretty good performance. Third, we also provide Bayesian interpretations of RNN which allow better uncertainty handling, which is useful for cosmological or any statistical analyses.

- **Can I use SuperNNova for my classification problem?**

Please do! But beware: you need to have a large amount of lightcurves (simulated or data) per type of event you are trying to classify, otherwise performance is pretty poor.

- **How can I use SuperNNova for my classification problem?**

It may require a little bit of code modification depending on your data. You can load data from SNANA formats (`.FITS` and `FITRES`, the latter is an ascii file) or `.csv` files (like the one from the Kaggle challenge, PlastiCC). Observations are grouped per night, so if you are looking for fast transients, you may need to create your own data pipeline or modify SuperNNova time grouping. Contact us if you have questions anais.moller@clermont.in2p3.fr and please report any issues!

6.2 Technical questions

- **What algorithms are available for classification?**

Currently we have a Baseline RNN and two Bayesian RNNs. The Bayesian RNNs are based on the work of Fortunato et al 2017 and Gal et Ghahramani 2015 and allow us to estimate prediction uncertainty. These algorithms require only raw lightcurve data. We have also a Random Forest classifier that relies lightcurve features. You can obtain these with fitters: an exponential that rises and falls or a type Ia supernova SALT2 fits.

- **Why is training slow ?**

If you have a GPU, you can activate training on GPU with the `--use_cuda` flag. Alternatively, you may select a smaller data fraction `--data_fraction 0.1` to train on a smaller set.

- **OSError: Unable to open file (unable to open file: name = '/home/snndump/processed/DES_database.h5'**

You have probably forgotten to set your `dump_dir` correctly. Provide the `--dump_dir` argument correctly.

- **Where do I find the model naming scheme?**

You can find it in `SuperNNova/utils/ExperimentSettings.py` under `model_name`. A start guide can be found in our *Quickstart guide (GitHub)*.

- **How do I change the directory where the data can be found?**

You can give add to your terminal command `--dump_dir foldername`. This folder should have the same structure as our data repositories (see *Data documentation*).

- **If I trained several models, is there a way to see a summary of the statistics?**

Yes, you need to call `python run.py --performance`. It will be created in `{dump_dir}/stats` as `summary_stats.csv`. It will compute various metrics which can be averaged over multiple random seeds. By default, this command will also generate all statistics (latex tables as well printout stats) and plots featured in our SuperNNova paper. To deactivate this, just comment in `run.py` the two lines below `# Stats and plots in paper`.

CHAPTER

SEVEN

DATA WALKTHROUGH

Recommended code organization structure:

```
├── snndump        (to save the data)
│
├── SuperNNova
    ├── supernnova
    ├── env
    ├── docs
    ├── tests
```

To build the database:

- Ensure you have raw data saved to `{raw_dir}/raw`
- The default settings assume the raw data and fits are saved to `snndump/raw`
- You can save the data in any folder, but you then have to specify the `dump_dir` with the `--dump_dir XXX` command.
- You can specify a different place where the raw data is using `--raw_dir XXX` command.
- You can specify a different place where the fits to data is using `--fits_dir XXX` command.

7.1 Activate the environment

Either use docker

```
cd env && python launch_docker.py (--use_cuda optional)
```

Or activate your conda environment

```
source activate <conda_env_name>
```

7.2 Creating a debugging database

Using command line: .. code:

```
python run.py --data --dump_dir tests/dump --raw_dir tests/raw --fits_dir tests/fits
```

- This creates a database for a very small subset of all available data
- This is intended for debugging purposes (training, validation can run very fast with this small database)
- The database is saved to the specified `tests/dump/processed`

Using yaml: .. code:

```
python run_yaml.py <yaml_file_with_config> --mode data
```

an example <yaml_file_with_config> is at `configs_yml`.

7.3 Creating a database

Using command line: .. code:

```
python run.py --data --dump_dir <path/to/full/database/> --raw_dir <path/to/raw/data/> --
↪fits_dir <path/to/fits/>
```

Using yaml: modify the configuration file .. code:

```
python run_yaml.py <yaml_file_with_config> --mode data
```

- You **DO NEED** to download the raw data for this database or point where your data is.
- This creates a database for all the available data with 80/10/10 train/validate/test splits.
- Splits can be changed using `--data_training` or `--data_testing` commands. For yaml just add `data_training: True` or `--data_testing: True`.
- The database is saved to the specified `dump_dir`, in the `processed` subfolder.
- There is no need to specify salt2fits file to make the dataset. It can be used if available but it is not needed `--fits_dir <empty/path/>`.
- Raw data can be in csv format with columns:
- `` DES_PHOT.csv ``: SNID,MJD, FLUXCAL, FLUXCALERR, FLT
- `` DES_HEAD.csv``: SNID, PEAKMJD, HOSTGAL_PHOTOZ, HOSTGAL_PHOTOZ_ERR, HOSTGAL_SPECZ, HOSTGAL_SPECZ_ERR, SIM_REDSHIFT_CMB, SIM_PEAKMAG_z, SIM_PEAKMAG_g, SIM_PEAKMAG_r, SIM_PEAKMAG_i, SNTYPE.

7.4 Creating a database for testing a trained model

This is how to create a database with only lightcurves to evaluate.

```
python run.py --dump_dir <path/to/save/database/> --data --data_testing  --raw_dir <path/
↪to/raw/data/>
```

Note that: - using `--data_testing` option will generate a 100% testing set (see below for more details). **Using command yaml:** modify the configuration file with `data_testing: True` and use the `--mode data`.

7.5 Creating a database with photometry limited to a time window

Photometric measurements may span over a larger time range than the one desired for classification. For example, a year of photometry is much larger than the usual SN timespan. Therefore, it may be desirable to just use a subset of this photometry (observed epochs cuts). To do so:

```
python run.py --dump_dir <path/to/save/database/> --data --raw_dir <path/to/raw/data/>  -
↪-photo_window_files <path/to/csv/with/peakMJD> --photo_window_var <name/of/variable/in/
↪csv/to/cut/on> --photo_window_min <negative/int/indicating/days/before/var> --photo_
↪window_max <positive/int/indicating/days/after/var>
```

7.6 Creating a database with different survey

The default filter set is the one from the Dark Energy Survey Supernova `g,r,i,z`. If you want to use your own survey, you'll need to specify your filters (Beware! as from 12/11/19 the input of possible combination of filters has been deprecated!).

```
python run.py --dump_dir <path/to/save/database/> --data --raw_dir <path/to/raw/data/>  -
↪-list_filters <your/filters>
```

e.g. ``–list_filters g r ``.

7.7 Using a different redshift label

The default redshift label is either `HOSTGAL_SPECZ/HOSTGAL_PHOTOZ` (with option `zspe/zpho`). If you want to use your own label, you'll need to specify it. Beware, this will override also `SIM_REDSHIFT_CMB` used for the title of plotted light-curves.

```
python run.py --dump_dir <path/to/save/database/> --data --raw_dir <path/to/raw/data/>  -
↪-redshift_label <your/label>
```

e.g. `--redshift_label REDSHIFT_FINAL`.

7.8 Masking photometry

The default is to use all available photometry for classification. However, we support masking photometric epochs with a power of two mask. Any combination of these power of two integers, and with other numbers, will be eliminated from the database.

```
python run.py --dump_dir <path/to/save/database/> --data --raw_dir <path/to/raw/data/>  --
↪-phot_reject <your/label> --phot_reject_list <list/to/reject>
```

e.g. `--phot_reject PHOTFLAG --phot_reject_list 8 16 32 64 128 256 512`.

7.9 Under the hood

7.9.1 Preparing data splits

We first compute the data splits:

- By default the HEAD FITS/csv files are analyzed to compute 80/10/10 train/test/val splits.
- You can change if the database contains 99.5/0.5/0.5 train/test/val splits using `--data_training` command.
- You can change if the database contains 0/0/100 train/test/val splits using `--data_testing` command. Beware, this option has other consequences.
- The splits are different for the salt/photometry datasets
- The splits are different depending on the classification target
- We downsample the dataset so that for a given classification task, all classes have the same cardinality
- The supernova/light-curve types supported can be changed using `--sntypes`. Default contains 7 classes. If a class is not given as input in `--sntypes`, it will be assigned to the last available tag. If a 'Ia' exists in provided `--sntypes`, this will be taken as tag 0 in data splits, else the first class will be used.

7.9.2 Preprocessing

We then pre-process each FITS/csv file

- Join column from header files
- Select columns that will be useful later on
- Compute SNID to tag each light curve
- Compute delta times between measures
- Removal of delimiter rows

7.9.3 Pivot

We then pivot each preprocessed file: we will group time-wise close observations on the same row and each row in the dataframe will show a value for each of the flux and flux error column

- All observations within 8 hours of each other are assigned the same MJD
- Results are cached with pickle for faster loading

7.9.4 HDF5

The processed database is saved to `dump_dir/processed` in HDF5 format for convenient use in the ML pipeline

The HDF5 file is organized as follows:

```
├── data                              (variable length array to store time series)
│
│
├── dataset_photometry_2classes       (0: train set, 1: valid set, 2: test set, -1: not␣
↪used)
├── dataset_photometry_7classes       (0: train set, 1: valid set, 2: test set, -1: not␣
↪used)
│
├── target_photometry_2classes        (integer between 0 and 1, included)
├── target_photometry_7classes        (integer between 0 and 6, included)
│
│
├── features                          (array of str: feature names to be used)
├── normalizations
    ├── FLUXCAL_g
         ├── min
         ├── mean                      Normalization coefficients for that feature
         ├── std
     ...
├── normalizations_global
    ├── FLUXCAL
         ├── min
         ├── mean                     Normalization coefficients for that feature
         ├── std                      In this scheme, the coefficients are shared between␣
↪fluxes and flux errors
    ...
│
├── SNID                              The ID of the lightcurve
├── PEAKMJD                           The MJD value at which a lightcurve reaches peak␣
↪light
├── SNTYPE                            The type of the lightcurve (120, 121...)
│
...                                   (Other metadata / features about lightcurves)
```

The features used for classification are the following:

- **FLUXCAL_g** (flux)
- **FLUXCAL_i** (flux)
- **FLUXCAL_r** (flux)

- **FLUXCAL_z** (flux)
- **FLUXCALERR_g** (flux error)
- **FLUXCALERR_i** (flux error)
- **FLUXCALERR_r** (flux error)
- **FLUXCALERR_z** (flux error)
- **delta_time** (time elapsed since previous observation in MJD)
- **HOSTGAL_PHOTOZ** (photometric redshift)
- **HOSTGAL_PHOTOZ_ERR** (photometric redshift error)
- **HOSTGAL_SPECZ** (spectroscopic redshift)
- **HOSTGAL_SPECZ_ERR** (spectroscopic redshift eror)
- **g** (boolean flag indicating which band is present at a specific time step)
- **gi** (boolean flag indicating which band is present at a specific time step)
- **gir** (boolean flag indicating which band is present at a specific time step)
- **girz** (boolean flag indicating which band is present at a specific time step)
- **giz** (boolean flag indicating which band is present at a specific time step)
- **gr** (boolean flag indicating which band is present at a specific time step)
- **grz** (boolean flag indicating which band is present at a specific time step)
- **gz** (boolean flag indicating which band is present at a specific time step)
- **i** (boolean flag indicating which band is present at a specific time step)
- **ir** (boolean flag indicating which band is present at a specific time step)
- **irz** (boolean flag indicating which band is present at a specific time step)
- **iz** (boolean flag indicating which band is present at a specific time step)
- **r** (boolean flag indicating which band is present at a specific time step)
- **rz** (boolean flag indicating which band is present at a specific time step)
- **z** (boolean flag indicating which band is present at a specific time step)

CHAPTER

EIGHT

DATA DOCUMENTATION

8.1 Dataset construction

8.2 PLASTICC Dataset construction

8.3 Data utilities

CHAPTER

NINE

HYPERPARAMETERS

9.1 General parameters

Argument	Type	Help
–seed	int	random seed to be used
–use_cuda	bool	Use GPU

9.2 Data parameters

Argument	Type	Help
–data	bool	if True, launch data creation
–dump_dir	str	path where data and models are dumped
–norm	str	Feature normalization used in training/validation: None, perfilter, global, cosmo, cosmo_quantile
–redshift	str	Host redshift used in training/validation: zpho, zspe or None
–source_data	str	Data source: photometry or salt
–no_overwrite	bool	If True, overwrite preprocessed dir when creating database
–data_fraction	float	Fraction of data to use
–over-ride_source_data	str	Change the source data (use saltfit or photometry)

9.3 Training parameters

Argument	Type	Help
–train_rnn	bool	Train RNN model
–train_rf	bool	Train RandomForest model
–monitor_interval	int	Validate every monitor_interval epochs–metrics

9.4 Validation Parameters

Argument	Type	Help
–validate_rnn	bool	Validate RNN model
–validate_rf	bool	Validate RandomForest model
–speed	bool	Run RNN speed classification benchmark
–calibration	bool	Evaluate model calibration
–performance	bool	Get performance metrics + plots
–metrics	bool	Compute performance metrics
–science_plots	bool	Plots of scientific interest
–model_files	bool	Path to model files
–prediction_files	bool	Path to prediction files
–metric_files	bool	Path to metric files

9.5 Visualization Parameters

Argument	Type	Help
–explore_lightcurves	bool	Plot a random selection of lightcurves
–plot_lcs	bool	Plot a random selection of lightcurves predictions
–plot_prediction_distribution	bool	Plot lcs and the histogram of probability for each class

9.6 RNN parameters

Argument	Type	Help
–cyclic	bool	Use cyclic learning rate
–cyclic_phases	list	Cyclic phases
–random_length	bool	Use random length sequences for training
–random_redshift	bool	If True, randomly set the spectroscopic redshift
–weight_decay	float	L2 decay on weights (for variational RNN)
–layer_type	str	Recurrent layer type. Choose lstm,gru,rnn
–model	str	Recurrent model type. Choose vanilla,variational,bayesian
–learning_rate	float	Learning rate
–nb_classes	int	Number of classification targets
–nb_epoch	int	Number of epoch
–batch_size	int	Batch size
–hidden_dim	int	Hidden layer dimension
–num_layers	int	Number of recurrent layers
–dropout	float	Dropout value
–bidirectional	bool	Use bidirectional models
–rnn_output_option	str	RNN output options. standard or mean
–pi	float	mixing coefficient for Bayes prior
–log_sigma1	float	Initialization parameter for BayesRNN layers
–log_sigma2	float	Initialization parameter for BayesRNN layers
–rho_scale_lower	float	Initialization parameter for BayesRNN layers
–rho_scale_upper	float	Initialization parameter for BayesRNN layers
–log_sigma1_output	float	Initialization parameter for BayesLinear output layers
–log_sigma2_output	float	Initialization parameter for BayesLinear output layers
–rho_scale_lower_output	float	Initialization parameter for BayesLinear output layers
–rho_scale_upper_output	float	Initialization parameter for BayesLinear output layers
–num_inference_samples	int	Number of samples to use for Bayesian inference
–mean_field_inference	bool	Use mean field inference for bayesian models

9.7 Random Forest parameters

Argument	Type	Help
–bootstrap	bool	Activate bootstrap when building trees
–min_samples_leaf	int	Minimum samples required to be a leaf node
–n_estimators	int	Number of trees
–min_samples_split	int	Min samples to create split
–criterion	str	Tree splitting criterion
–max_features	int	Max features per tree
–max_depth	int	Max tree depth

CHAPTER

TEN

EXPERIMENT SETTINGS

CHAPTER

ELEVEN

TRAINING WALKTHROUGH

11.1 Activate the environment

Either use docker

```
cd env && python launch_docker.py (--use_cuda optional)
```

Or activate your conda environment

```
source activate <conda_env_name>
```

11.1.1 Training an RNN model

Using command line: .. code:

```
python run.py --data --dump_dir /path/to/your/dump/dir # build the data
python run.py --train_rnn --dump_dir /path/to/your/dump/dir # train and validate
```

Using Yaml: .. code:

```
python run_yaml.py <yaml_file_with_config> --mode train_rnn
```

an example <yaml_file_with_config> is at configs_yml.

This will:

- Train an RNN classifier
- All outputs are dumped to /path/to/your/dump/dir/models/vanilla_S_0_CLF_2_R_None_saltfit_DF_1.0_N_global_lstm_32x2_0.05_128_True_mean
- Save the trained classifier: vanilla_S_0_CLF_2_R_None_saltfit_DF_1.0_N_global_lstm_32x2_0.05_128_True_mean.pt
- Make predictions on a test set: PRED_DES_vanilla_S_0_CLF_2_R_None_saltfit_DF_1.0_N_global_lstm_32x2_0.05_128_True_mean.pt.pickle
- Compute metrics on the test: METRICS_DES_vanilla_S_0_CLF_2_R_None_saltfit_DF_1.0_N_global_lstm_32x2_0.05_128_True_mean.pt.pickle
- Save loss curves: train_and_val_loss_vanilla_S_0_CLF_2_R_None_saltfit_DF_1.0_N_global_lstm_32x2_0.05_128_True_mean.png
- Save training statistics: training_log.json

11.1.2 Training an RNN model with different normalizations

The data for training and validation can be normalized for better performance. Currrently the options for `--norm` are `none, global, perfilter, cosmo, cosmo_quantile`. The default normalization is `global`.

For `global, perfilter` normalizations, features (f) are first log transformed and then scaled. The log transform (fl) uses the minimum value of the feature min(f) and a constant (epsilon) to center the distribution in zero as follows: fl = log (min(f) + f + epsilon). Using the mean and standard deviation of the log transform (mu,sigma(fl)), standard scaling is applied: f^ = (fl mu(fl))/sigma(fl). In the "global" scheme, the minimum, mean and standard deviation are computed over all fluxes (resp. all errors). In the "per-filter" scheme, they are computed for each filter.

When using `--redshift` for classification, we suggest to use either `cosmo,cosmo_quantile` norms. These normalizations blur the distance information that SNe Ia provide with apparent flux which together with redshift information may bias the classification for cosmology. For this, light-curves are normalized to a flux ~1 using either the maximum flux at any filter (`cosmo`) or the 99 quantile of the flux distribution (`cosmo_quantile`). The latter is mroe robust against outliers.

11.1.3 Training a randomforest model (paper branch)

```
python run.py --data --dump_dir /path/to/your/dump/dir # build the data
python run.py --train_rf --dump_dir /path/to/your/dump/dir # train and validate
```

This will:

- Train a randomforest classifier
- All outputs are dumped to `/path/to/your/dump/dir/models/randomforest_S_0_CLF_2_R_None_saltfit_DF_1.0_N_global`
- Save the trained classifier: `randomforest_S_0_CLF_2_R_None_saltfit_DF_1.0_N_global.pickle`
- Make predictions on a test set: `PRED_DES_randomforest_S_0_CLF_2_R_None_saltfit_DF_1.0_N_global.pickle`
- Compute metrics on the test: `METRICS_DES_randomforest_S_0_CLF_2_R_None_saltfit_DF_1.0_N_global.pickle`

Beware: RF is not currently supported for Yaml runs.

CHAPTER

TWELVE

TRAINING DOCUMENTATION

12.1 RNN models

12.2 RNN trainer

12.3 RandomForest trainer

12.4 Training utilities

CHAPTER

THIRTEEN

VALIDATION WALKTHROUGH

13.1 Activate the environment

Either use docker

```
cd env && python launch_docker.py (--use_cuda optional)
```

Or activate your conda environment

```
source activate <conda_env_name>
```

13.2 Validation

Assuming a database has been created and models have been trained, a model can be validated as follows:

Using command line: .. code:

```
python run.py --validate_rnn --dump_dir /path/to/dump_dir
python run.py --validate_rnn --dump_dir /path/to/dump_dir
```

Using Yaml: .. code:

```
python run_yaml.py <yaml_file_with_config> --mode validate_rnn
```

an example <yaml_file_with_config> is at configs_yml.

In that case, the model corresponding to the command line arguments will be loaded and validated. Output will be written in dump_dir/models/yourmodelname/.

Alternatively, one or more model files can be specified

```
python run.py --validate_rnn --dump_dir /path/to/dump_dir --model_files /path/to/model/
↪file(s)
python run.py --validate_rnn --dump_dir /path/to/dump_dir --model_files /path/to/model/
↪file(s)
```

In that case, validation will be carried out for each of the models specified by the model files. This will use the database in dump_dir/processed directory.

This will:

- Make predictions on a test set (saved to a file with the PRED_ prefix)

- Compute metrics on the test (saved to a file with the METRICS_ prefix)
- All results are dumped in the same folder as the folder where the trained model was dumped

To make predictions on an independent database than the one used to train a given model

```
python run.py --dump_dir  /path/to/dump_dir --validate_rnn --model_files path/to/
↪modelfile/modelfile.pt
```

In this case it will run the model provided in `model_files` with the normalization of the model on the database available in `dump_dir/processed`. Predictions will be saved in `dump_dir/models/modelname/`.

13.2.1 Predictions format

For a binary classification task, predictions files contain the following columns:

```
all_class0            float32  - probability of classifying complete light-curves as --
↪sntype [0] (usually Ia)
all_class1            float32  - probability of classifying complete light-curves as --
↪sntype [1:] (usually nonIas)
PEAKMJD-2_class0      float32  - probability of classifying light-curves up to 2 days␣
↪before maximum as --sntype [0] (usually Ia)
PEAKMJD-2_class1      float32  - probability of classifying light-curves up to 2 days␣
↪before maximum as  --sntype [1:] (usually nonIas)
PEAKMJD-1_class0      float32  - up to one day before maximum light
PEAKMJD-1_class1      float32
PEAKMJD_class0        float32  - up to maximum light lightcurves
PEAKMJD_class1        float32
PEAKMJD+1_class0      float32  - one day post maximum lightcurves
PEAKMJD+1_class1      float32
PEAKMJD+2_class0      float32  - two days post maximum lightcurves
PEAKMJD+2_class1      float32
all_random_class0     float32  - Out-of-distribution: probability of classifying␣
↪randomly generated complete lightcurves as --sntype [0]
all_random_class1     float32
all_reverse_class0    float32  - Out-of-distribution: probability of classifying time␣
↪reversed complete lightcurves as --sntype [0]
all_reverse_class1    float32
all_shuffle_class0    float32  - Out-of-distribution: probability of classifying␣
↪shuffled complete lightcurves (permutations of time-series) as --sntype [0]
all_shuffle_class1    float32
all_sin_class0        float32  - Out-of-distribution: probability of classifying␣
↪sinusoidal complete lightcurves (permutations of time-series) as --sntype [0]
all_sin_class1        float32
target                  int64  - Type of the supernova, simulated class.
SNID                    int64  - ID number of the light-curve
```

these columns rely on maximum light information and target (original type) from simulations. Out-of-distribution classifications are done on the fly. Bayesian Networks (variational and Bayes by Backprop) have an entry for each probability distribution sampling, to get the mean and std of the classification read the `_aggregated.pickle` file.

13.3 RNN speed

Run RNN classification speed benchmark as follows

```
python run.py --data --dump_dir /path/to/dump_dir  # create database
python run.py --speed --dump_dir /path/to/dump_dir
```

This will create `tests/dump/stats/rnn_speed.csv` showing the classification throughput of RNN models.

13.4 Calibration

Assuming a database has been created and models have been trained, evaluate classifier calibration as follows:

```
python run.py --calibration --dump_dir /path/to/dump_dir --metric_files /path/to/metric_
↪file
```

This will output a figure in `path/to/dump_dir/figures` showing how well a given model is calibrated. A metric file looks like this: `METRICS_{model_name}.pickle`. For instance: `METRICS_DES_vanilla_S_0_CLF_2_R_None_saltfit_DF_1.0_N_global_lstm_32x2_0.05_128_True_mean.pickle` Multiple metric files can be specified, the results will be charted on the same graph.

13.5 Science plots

Assuming a database has been created and models have been trained, how some graphs of scientific interest:

```
python run.py --science_plots --dump_dir /path/to/dump_dir --prediction_files /path/to/
↪prediction_file
```

This will output figures in `path/to/dump_dir/figures` showing various plots of interest: Hubble residuals, purity vs redshift etc. A prediction file looks like this: `PRED_{model_name}.pickle`. For instance: `PRED_DES_vanilla_S_0_CLF_2_R_None_saltfit_DF_1.0_N_global_lstm_32x2_0.05_128_True_mean.pickle`

13.6 Performance metrics

Assuming a database has been created and models have been trained, compute performance metrics

```
python run.py --performance --dump_dir /path/to/dump_dir
```

This will output a csv file in `path/to/dump_dir/stats`, which aggregates various performance metrics for each model that has been trained and for which a `METRICS` file has been created.

CHAPTER

FOURTEEN

VALIDATION DOCUMENTATION

14.1 Validation RandomForest

14.2 Validation RNN

14.3 Metrics

CHAPTER

FIFTEEN

VISUALIZATION WALKTHROUGH

15.1 Activate the environment

Either use docker

```
cd env && python launch_docker.py (--use_cuda optional)
```

Or activate your conda environment

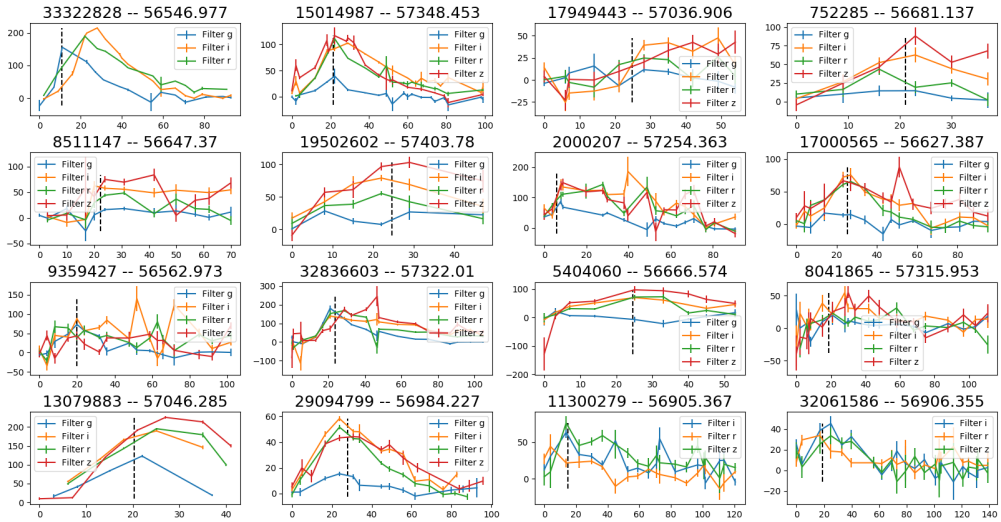
```
source activate <conda_env_name>
```

15.2 Exploring the dataset

Using command line: .. code:

```
python run.py --data --dump_dir tests/dump  # build the database
python run.py --explore_lightcurves --dump_dir tests/dump
```

Outputs: `.png` files in the `tests/dump/explore` folder. You should obtain something that looks like this:



15.3 Predictions as a function of time

Assuming you have a trained model stored under `tests/dump/models/vanilla_S_0_CLF_2_R_None_saltfit_DF_1.0_N_global_lstm_32x2_0.05_128_True_mean` and that you have already created the database as above:

Using command line: .. code:

```
python run.py --plot_lcs --dump_dir tests/dump --model_files tests/dump/models/vanilla_S_
↪0_CLF_2_R_None_saltfit_DF_1.0_N_global_lstm_32x2_0.05_128_True_mean/vanilla_S_0_CLF_2_
↪R_None_saltfit_DF_1.0_N_global_lstm_32x2_0.05_128_True_mean.pt
```

Using Yaml: .. code:

```
python run_yaml.py <yaml_file_with_config> --mode plot_lcs
```

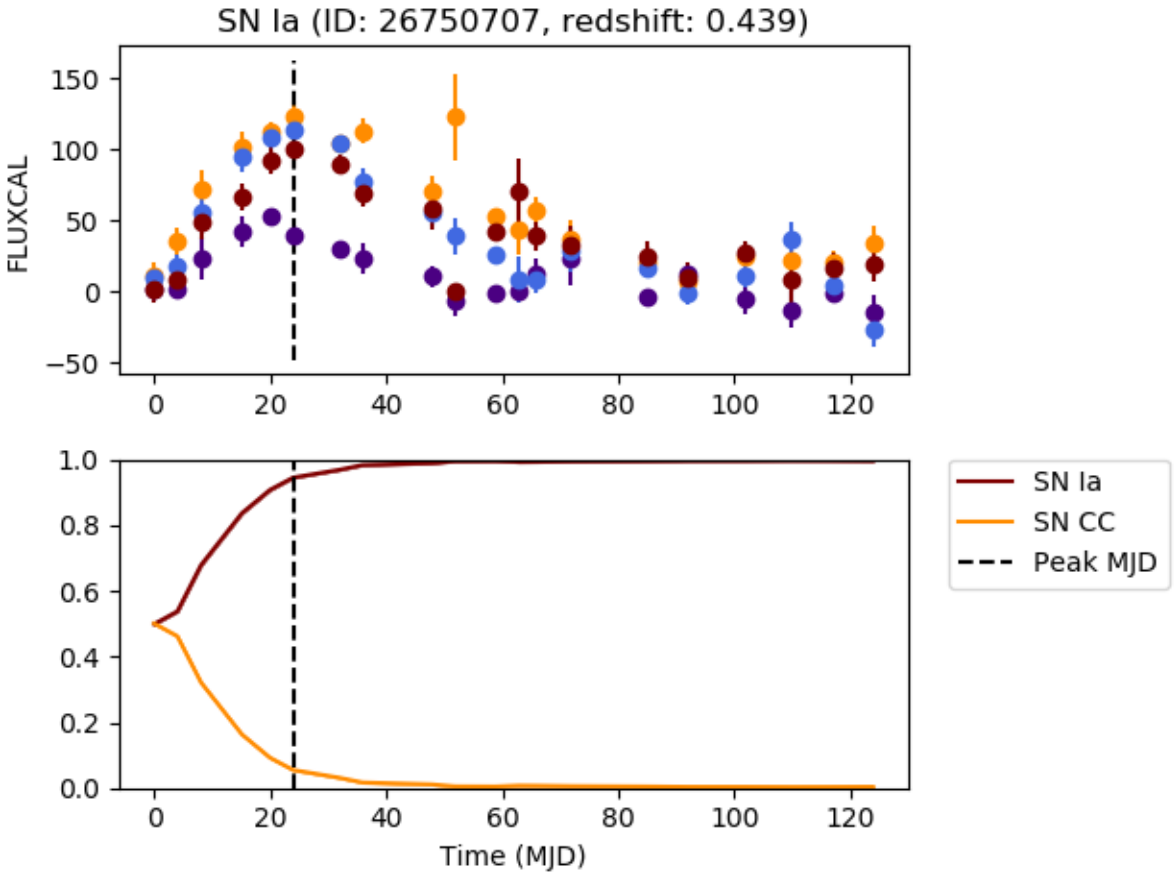
an example <yaml_file_with_config> is at `configs_yml`.

Outputs: a figure folder under `tests/dump/lightcurves/vanilla_S_0_CLF_2_R_None_saltfit_DF_1.0_N_global_lstm_32x2_0.05_128_True_mean`.

This folder contains the plot of several random lightcurves and the predictions made by the neural network referred to by the `model_files` argument.

If you want to plot a selection of lightcurves you can add `--plot_file <filename.csv>` which contains a column `SNID` with the ids requested to be plotted.

Below is a sample plot:



15.4 Predictions + uncertainty for bayesian models

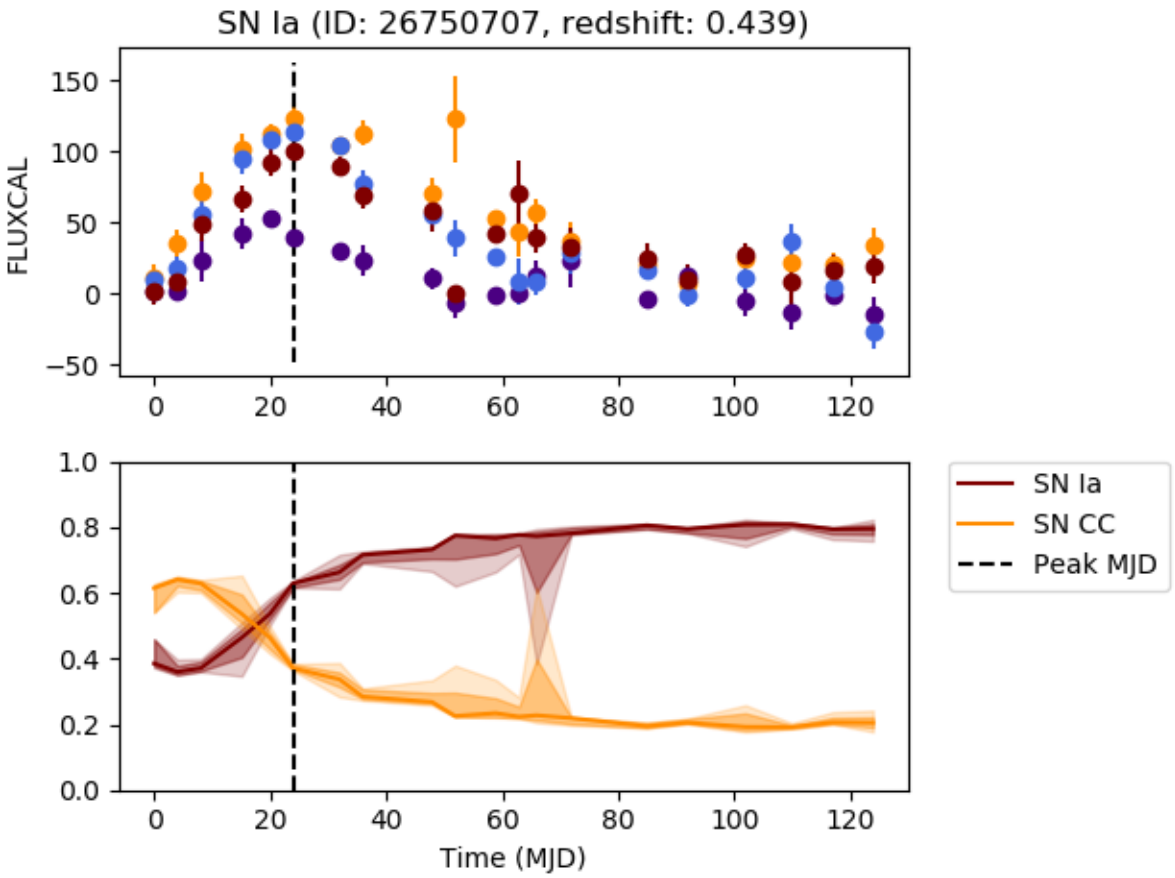
Assuming you have a variational RNN model stored under `tests/dump/models/variational_S_0_CLF_2_R_None_saltfit_DF_1.0_N_global_lstm_32x2_0.05_128_True_mean_WD_1e-07` and that you have already created the database as above:

```
python run.py --plot_lcs --dump_dir tests/dump --model_files tests/dump/models/
↪variational_S_0_CLF_2_R_None_saltfit_DF_1.0_N_global_lstm_32x2_0.05_128_True_mean_WD_
↪1e-07/variational_S_0_CLF_2_R_None_saltfit_DF_1.0_N_global_lstm_32x2_0.05_128_True_
↪mean_WD_1e-07.pt
```

Outputs: a figure folder under `tests/dump/lightcurves/variational_S_0_CLF_2_R_None_saltfit_DF_1.0_N_global_lstm_32x2_0.05_128_True_mean_WD_1e-07`.

This folder contains the plot of several lightcurves and the predictions made by the neural network referred to by the `model_files` argument. Several predictions are sampled at each timestep and the prediction contours at 68% and 94% are shown.

Below is a sample plot:



15.5 Predictions from multiple models

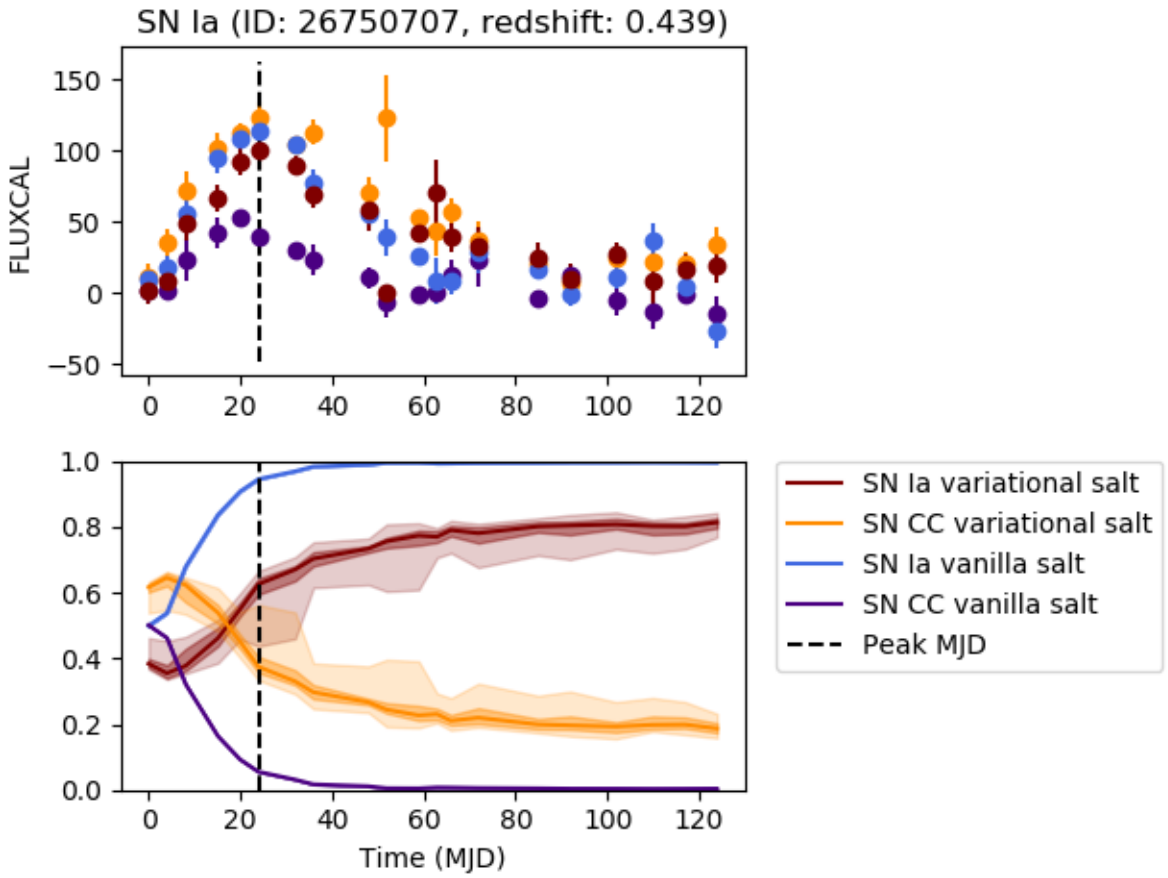
To compare the predictions from multiple models, simply call the above, while providing multiple `model_files`

```
python run.py --plot_lcs --dump_dir tests/dump --model_files tests/dump/models/
↪variational_S_0_CLF_2_R_None_saltfit_DF_1.0_N_global_lstm_32x2_0.05_128_True_mean_WD_
↪1e-07/variational_S_0_CLF_2_R_None_saltfit_DF_1.0_N_global_lstm_32x2_0.05_128_True_
↪mean_WD_1e-07.pt tests/dump/models/vanilla_S_0_CLF_2_R_None_saltfit_DF_1.0_N_global_
↪lstm_32x2_0.05_128_True_mean/vanilla_S_0_CLF_2_R_None_saltfit_DF_1.0_N_global_lstm_
↪32x2_0.05_128_True_mean.pt
```

Outputs: a figure folder under `tests/dump/figures/multimodel_early_prediction`.

This folder contains the plot of several lightcurves and the predictions made by the neural networks referred to by the `model_files` argument.

Below is a sample plot:



15.6 Science plots

The plots of the paper can be reproduced by running in the `paper branch`:

```
python run_paper.py
```

CHAPTER

SIXTEEN

VISUALIZATION DOCUMENTATION

16.1 Dataset exploration

16.2 Plotting predictions

CHAPTER

SEVENTEEN

PAPER REPRODUCTION WALKTHROUGH

17.1 Reproducing the models

To reproduce the stats on the paper you need first to run all the models

```
python run_paper.py
```

With a GPU and a `--batch_size = 128` (default) this takes around two weeks. If you increase `batch_size` it may be reduced to a couple of days but performance can be slightly reduced.

17.2 Reproducing the stats and plots

Summary statistics for all trained models and a printout with the stats and plots used in the paper are produced by:

```
python run.py --performance
```

Summary statistics will be found in `snndump/stats/summary_stats.csv`. Statistics used in the paper are printed out and latex tables created in `snndump/latex/`. Plots and figures are found in `snndump/figures/` and `snndump/lightcurves/`.

To obtain summary statistics only, comment in the two lines after `# Stats and plots in paper` in `run.py`.

To obtain stats only, comment the plotting function at `SuperNNova/supernnova/paper/paper_thread.py` by changing `SuperNNova_stats_and_plots_thread(df_stats, settings, plots=False)`.

CHAPTER

EIGHTEEN

PAPER REPRODUCTION

18.1 Thread

18.2 Metrics

18.3 Plots