astroNN Documentation

Release 1.0.1

Henry Leung

Mar 06, 2019

Contents

astroNN is a python package to do various kinds of neural networks with targeted application in astronomy by using Keras as model and training prototyping, but at the same time take advantage of Tensorflow's flexibility.

For non-astronomy applications, astroNN contains custom loss functions and layers which are compatible with Tensorflow or Keras with Tensorflow backend. The custom loss functions mostly designed to deal with incomplete labels. astroNN contains demo for implementing Bayesian Neural Net with Dropout Variational Inference in which you can get reasonable uncertainty estimation and other neural nets.

For astronomy applications, astroNN contains some tools to deal with APOGEE, Gaia and LAMOST data. astroNN is mainly designed to apply neural nets on APOGEE spectra analysis and predicting luminosity from spectra using data from Gaia parallax with reasonable uncertainty from Bayesian Neural Net. Generally, astroNN can handle 2D and 2D colored images too. Currently astroNN is a python package being developed by the main author to facilitate his research project on deep learning application in stellar and galactic astronomy using SDSS APOGEE, Gaia and LAMOST data.

For learning purpose, astroNN includes a deep learning toy dataset for astronomer - /galaxy10.

CHAPTER 1

Getting Started

astroNN is developed on GitHub. You can download astroNN from its Github.

But the easiest way to install is via `pip`: astroNN on Python PyPI

```
$ pip install astroNN
```

For the latest version, you can clone the latest commit of astroNN from github

```
$ git clone --depth=1 https://github.com/henrysky/astroNN
```

and run the following command to install after you open a command line window in the package folder:

```
$ python setup.py install
```

CHAPTER 2

Indices, tables and astroNN structure

- genindex
- modindex
- search

```
astroNN/
├── apogee/
│   ├── apogee_shared.py [shared codes across apogee module]
│   ├── chips.py [functions to deal with apogee detectors and spectra]
│   ├── downloader.py [functions to downlaod apogee data]
│   └── plotting.py [functions to plot apogee data]
├── data/
│   └── ... [multiple pre-compiled data in numpy format]
├── datasets/
│   ├── apogee_distances.py
│   ├── apogee_rc.py
│   ├── apokasc.py
│   ├── galaxy10.py [astroNN's galaxy10 related codes]
│   ├── h5.py
│   └── xmatch.py [coordinates cross matching]
├── gaia/
│   ├── downloader.py [functions to downlaod gaia data]
│   └── gaia_shared.py [function related to astrometry and magnitude]
├── lamost/
│   ├── chips.py [functions to deal with lamost detectors and spectra]
│   └── lamost_shared.py [shared codes across lamost module]
├── models/ [contains neural network models]
│   └── ... [NN models codes and modules]
├── nn/
│   ├── callbacks.py [Keras's callbacks]
│   ├── layers.py [Tensorflow layers]
│   ├── losses.py [Tensorflow losses]
│   ├── metrics.py [Tensorflow metrics]
```

(continues on next page)

(continued from previous page)

```
    └──  numpy.py [handy numpy implementation of NN tools]
└── shared/ [shared codes across modules]
```

CHAPTER 3

Datasets

- /galaxy10

CHAPTER 4

Basics of astroNN

4.1 Getting Started

astroNN is developed on GitHub. You can download astroNN from its Github.

But the easiest way to install is via `pip`: astroNN on Python PyPI

```
$ pip install astroNN
```

For latest version, you can clone the latest commit of astroNN from github

```
$ git clone --depth=1 https://github.com/henrysky/astroNN
```

and run the following command to install after you open a command line window in the package folder:

```
$ python setup.py install
```

4.1.1 Prerequisites

Latest version of Anaconda is recommended, but generally the use of Anaconda is still highly recommended

```
Python 3.6 or above
Tensorflow OR Tensorflow-gpu (the latest version is recommended)
Tensorflow-Probability (the latest version is recommended)
Keras (Optional but the latest Keras version is recommended, Must be configured␣
↪Tensorflow as backends)
CUDA and CuDNN (only necessary for Tensorflow-gpu)
graphviz and pydot are required to plot the model architecture
scikit-learn, tqdm and astroquery required for some basic astroNN function
```

Since Tensorflow, Tensorflow-Probability and Keras are rapidly developing packages and astroNN heavily depends on Tensorflow. The support policy of astroNN to these packages is only the last 2 official versions are supported (i.e.

the latest and the previous version are included in test suite). Generally the latest version of Tensorflow, Tensorflow-Probability and optional Keras is recommended. The current supporting status (i.e. included in test cases) are

```
Tensorflow OR Tensorflow-gpu 1.13.1 (correspond to Tensorflow-Probability 0.6.0) without Keras
Tensorflow OR Tensorflow-gpu 1.13.1 (correspond to Tensorflow-Probability 0.6.0) with Keras 2.2.4
Tensorflow OR Tensorflow-gpu 1.12.0 (correspond to Tensorflow-Probability 0.5.0) with Keras 2.2.0
```

Note: Although Tensorflow/Tensorflow-gpu 1.12.0 is currently supported. But due to a bug: https://github.com/tensorflow/tensorflow/issues/22952 You have to patch a line in Tensorflow installation.

For the file under your python installation `site-packages/tensorflow/python/keras/engine/training_generator.py` Change Line 354 from

```
model._make_test_function()
```

to

```
model._make_predict_function()
```

A patched `training_generator.py` file is available at https://github.com/henrysky/astroNN/blob/master/travis_tf_1_12_patch.py

For instruction on how to install Tensorflow, please refers to their official website Installing TensorFlow

Although Keras is optional, but it is highly recommended. For instruction on how to install Keras, please refers to their official website Installing Keras

If you install *tensorflow* instead of *tensorflow-gpu*, Tensorflow will run on CPU. Currently official Tensorflow python wheels do not compiled with AVX2 - sets of CPU instruction extensions that can speed up calculation on CPU. If you are using *tensorflow* which runs on CPU only or want to use latest CUDA/CuDNN , you should consider to download High Performance Tensorflow MacOS build for MacOS, Or High Performance Tensorflow Windows build for Windows.

Recommended system requirement:

```
64-bits operating system
CPU which supports AVX2 (List of CPUs: https://en.wikipedia.org/wiki/Advanced_Vector_Extensions#CPUs_with_AVX2)
8GB RAM or above
NVIDIA Graphics card (Optional, GTX900 series or above)
(If using NVIDIA GPU): At least 2GB VRAM on GPU
```

Note: Multi-GPU, Intel/AMD graphics are not supported. Only Windows and Linux are officially supported by Tensorflow-GPU with compatible NVIDIA graphics

4.1.2 Basic FAQ

My hardware or software cannot meet the prerequisites, what should I do?

The hardware and software requirement is just an estimation. It is entirely possible to run astroNN without those requirement. But generally, python 3.6 or above (as Tensorflow only supports py36 or above) and mid-to-high end

hardware.

Can I contribute to astroNN?

You can contact me (Henry: henrysky.leung [at] mail.utoronto.ca) or refer to *Contributor and Issue Reporting guide*.

I have found a bug in astorNN

Please try to use the latest commit of astroNN. If the issue persists, please report to https://github.com/henrysky/astroNN/issues

I keep receiving warnings on APOGEE and Gaia environment variables

If you are not dealing with APOGEE or Gaia data, please ignore those warnings. If error raised to prevent you to use some of astroNN functionality, please report it as a bug to https://github.com/henrysky/astroNN/issues

If you don't want those warnings to be shown again, go to astroNN's configuration file and set `environmentvariablewarning` to `False`

I have installed *pydot* and *graphviz* but still fail to plot the model

if you are encountering this issue, please uninstall both `pydot` and `graphviz` and run the following command

```
$ pip install pydot
$ conda install graphviz
```

Then if you are using Mac, run the following command

```
$ brew install graphviz
```

If you are using Windows, go to https://graphviz.gitlab.io/_pages/Download/Download_windows.html to download the Windows package and add the package to the PATH environment variable.

4.1.3 Configuration File

astroNN configuration file is located at `~/.astroNN/config.ini` which contains a few astroNN settings.

Currently, the default configuration file should look like this

```
[Basics]
magicnumber = -9999.0
multiprocessing_generator = False
environmentvariablewarning = True
tensorflow_keras = auto

[NeuralNet]
custommodelpath = None
cpufallback = False
gpu_mem_ratio = True
```

`magicnumber` refers to the Magic Number which representing missing labels/data, default is -9999. Please do not change this value if you rely on APOGEE data.

`multiprocessing_generator` refers to whether enable multiprocessing in astroNN data generator. Default is False except on Linux and MacOS.

`environmentvariablewarning` refers to whether you will be warned about not setting APOGEE and Gaia environment variable.

`tensorflow_keras` refers to whether use *keras* or *tensorflow.keras*. Default option is `auto` to let astroNN to decide (*keras* always be considered first), `tensorflow` to force it to use *tensorflow.keras* or `keras` to force it to use *keras*

`custommodelpath` refers to a list of custom models, path to the folder containing custom model (.py files), multiple paths can be separated by `;`. Default value is *None* means no path. Or for example: `/users/astroNN/custom_models/;/local/some_other_custom_models/`

`cpufallback` refers to whether force to use CPU. No effect if you are using tensorflow instead of tensorflow-gpu

`gpu_mem_ratio` refers to GPU management. Set `True` to dynamically allocate memory which is astroNN default or enter a float between 0 and 1 to set the maximum ratio of GPU memory to use or set `None` to let Tensorflow pre-occupy all of available GPU memory which is a designed default behavior from Tensorflow.

For whatever reason if you want to reset the configure file:

```
from astroNN.config import config_path

# astroNN will reset the config file if the flag = 2
config_path(flag=2)
```

4.1.4 Folder Structure for astroNN, APOGEE, Gaia and LAMOST data

This code depends on environment variables and folders for APOGEE, Gaia and LAMOST data. The environment variables are

- `SDSS_LOCAL_SAS_MIRROR`: top-level directory that will be used to (selectively) mirror the SDSS Science Archive Server (SAS)
- `GAIA_TOOLS_DATA`: top-level directory under which the Gaia data will be stored.
- `LASMOT_DR5_DATA`: top-level directory under which the LASMOST DR5 data will be stored.

How to set environment variable on different operating system: Guide here

```
$SDSS_LOCAL_SAS_MIRROR/
├── dr14/
│   ├── apogee/spectro/redux/r8/stars/
│   │   ├── apo25m/
│   │   │   ├── 4102/
│   │   │   │   ├── apStar-r8-2M21353892+4229507.fits
│   │   │   │   ├── apStar-r8-******************.fits
│   │   │   │   └── ****/
│   │   ├── apo1m/
│   │   │   ├── hip/
│   │   │   │   ├── apStar-r8-2M00003088+5933348.fits
│   │   │   │   ├── apStar-r8-******************.fits
│   │   │   │   └── ***/
│   │   ├── l31c/l31c.2/
│   │   │   ├── allStar-l30e.2.fits
```

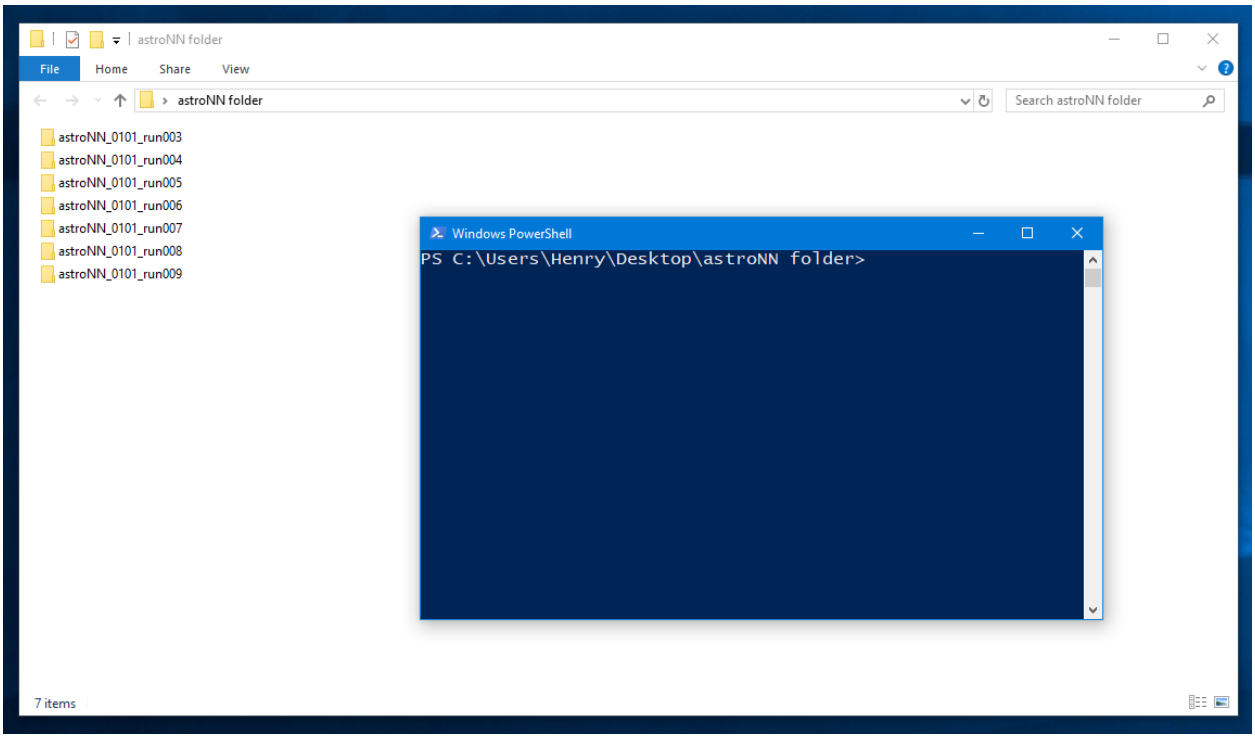
(continues on next page)

(continued from previous page)

```
│   │   │   ├── allVisit-l30e.2.fits
│   │   │   ├── 4102/
│   │   │   │   ├── aspcapStar-r8-l30e.2-2M21353892+4229507.fits
│   │   │   │   ├── aspcapStar-r8-l30e.2-**********+*******.fits
│   │   │   │   └── ****/
│   │   │   └── Cannon/
│   │   │       └── allStarCannon-l31c.2.fits
└── dr13/
    └── *similar to dr14 above/*


$GAIA_TOOLS_DATA/
└── Gaia/
    ├── gdr1/tgas_source/fits/
    │   ├── TgasSource_000-000-000.fits
    │   ├── TgasSource_000-000-001.fits
    │   └── ***.fits
    └── gdr2/gaia_source_with_rv/fits/
        ├── GaiaSource_2851858288640_1584379458008952960.fits
        ├── GaiaSource_1584380076484244352_2200921635402776448.fits
        └── ***.fits

$LASMOT_DR5_DATA/
└── DR5/
    ├── LAMO5_2MS_AP9_SD14_UC4_PS1_AW_Carlin_M.fits
    ├── 20111024
    │   ├── F5902
    │   │   ├──spec-55859-F5902_sp01-001.fits.gz
    │   │   └── ****.fits.gz
    │   └── ***/
    ├── 20111025
    │   ├── B6001
    │   │   ├──spec-55860-B6001_sp01-001.fits.gz
    │   │   └── ****.fits.gz
    │   └── ***/
    └── ***/
```

Note: The APOGEE and Gaia folder structure should be consistent with APOGEE and gaia_tools python package by Jo Bovy, tools for dealing with APOGEE and Gaia data

A dedicated project folder is recommended to run astroNN, always run astroNN under the root of project folder. So that astroNN will always create folder for every neural network you run under the same place. Just as below



4.2 Contributor and Issue Reporting guide

When contributing to this repository, please first discuss the big changes you wish to make via opening issue, email, or any other method with the maintainers of this repository.

4.2.1 Submitting bug reports and feature requests

Bug reports and feature requests should be submitted by creating an issue on https://github.com/henrysky/astroNN

4.2.2 Pull Request

This is a general guideline to make pull request (PR).

1. Go to https://github.com/henrysky/astroNN, click the `Fork` button.
2. Download your own astroNN fork to your computer by `$git clone https://github.com/your_username/astroNN`
3. Create a new branch with a short simple name that represents the change you want to make
4. Make commits locally in that new branch, and push to your own astroNN fork repository
5. Create a pull request by clicking the `New pull request` button.

4.2.3 New Model Proposal guide

astroNN acts as a platform to share astronomy-oriented neural networks, so you are welcome to do so.

To add new models:

- Import your model in `astroNN\models\__init__.py` and add the model class name to `__all__`
- Add a documentation page for the new model and add link it appropriately in `docs\source\index.rst`
- Add the new model to the tree diagram and API under appropriate class in `docs\souce\neuralnets\basic_usage.rst`
- Add the new model to the release history in `docs\source\history.rst`

If your new model is proposed along with a paper, add your model to the test suite in `tests\test_paper_models.rst` just to make sure your model works fine against future changes in astroNN.

4.2.4 Possible New Features and Improvement in the future

Anticipating Tensorflow 2.0

- Drop Keras support as Keras is deeply integrated in Tensorflow 2.0?
- Need to keep track of deprecation warning in current Tensorflow 1.1x series to better prepare for tf2.0

GPU performance related issues

- Data reduction pipeline on GPU?
- Multiple GPU support!

Neural Network related issues

- Currently the Bayesian NN models only use Dropout VI, maybe introduce more methods especially from TF-Probability
- Have some nice VAE or GAN thing, maybe on spectroscopic data first

4.3 History

4.3.1 v1.1 series

v1.1.0 (xx xxx 20xx)

Pending

4.3.2 v1.0 series

v1.0.1 (5 March 2019)

This release mainly targeted to the paper `Simultaneous calibration of spectro-photometric distances and the Gaia DR2 parallax zero-point offset with deep learning` available at [arXiv:1902.08634] [ADS]

Documentation for this version is available at https://astronn.readthedocs.io/en/v1.0.1/

New features:

- Better and faster with IPython tab auto-completion
- Added models : `ApogeeDR14GaiaDR2BCNN`

Improvement:

- Improved data pipeline to generate data for NNs

Breaking Changes:

- Tested with Tensorflow 1.11.0/1.12.0/1.13.1 and Keras 2.2.0/2.2.4

v1.0.0 (16 August 2018)

This is the first release of astroNN. This release mainly targeted to the paper `Deep learning of multi-element abundances from high-resolution spectroscopic data` available at [arXiv:1804.08622] [ADS]

Documentation for this version is available at https://astronn.readthedocs.io/en/v1.0.0/

New features:

- Initial Release!!

Breaking Changes:

- Tested with Tensorflow 1.8.0/1.9.0 and Keras 2.2.0/2.2.2
- Python 3.6 or above only

4.3.3 v0.0 series

v0.0.0 (13 October 2017)

First commit of astroNN on Github!!!

4.4 Loss Functions and Metrics - astroNN.nn.losses

astroNN provides modified loss functions which are capable to deal with incomplete labels which are represented by `magicnumber` in astroNN configuration file or `Magic Number` in equations below. Since they are built on Tensorflow and follows Keras API requirement, all astroNN loss functions are fully compatible with Keras with Tensorflow backend, as well as directly be imported and used with Tensorflow, for most loss functions, the first argument is ground truth tensor and the second argument is prediction tensor from neural network.

Note: Always make sure when you are normalizing your data, keep the magic number as magic number. If you use astroNN normalizer, astroNN will take care of that.

Here are some explanations on variables in the following loss functions:

y_i means the ground truth labels, always represented by python variable `y_true` in astroNN

$\hat{y_i}$ means the prediction from neural network, always represented by python variable `y_pred` in astroNN

4.4.1 Correction Term for Magic Number

`astroNN.nn.`**`magic_correction_term`**(*y_true*)

Calculate a correction term to prevent the loss being lowered by magic_num

Parameters **`y_true`** (*`tf.Tensor`*) – Ground Truth

Returns Correction Term

Return type tf.Tensor

History

2018-Jan-30 - Written - Henry Leung (University of Toronto)
2018-Feb-17 - Updated - Henry Leung (University of Toronto)

Since astroNN deals with magic number by assuming the prediction from neural network for those ground truth with Magic Number is right, so we need a correction term.

The correction term in astroNN is defined by the following equation and we call the equation $\mathcal{F}_{correction}$

$$\mathcal{F}_{correction} = \frac{\text{Non-Magic Number Count} + \text{Magic Number Count}}{\text{Non Magic Number Count}}$$

In case of no labels with Magic Number is presented, $\mathcal{F}_{correction}$ will equal to 1

4.4.2 Mean Squared Error

`astroNN.nn.losses.`**`mean_squared_error`**(*y_true*, *y_pred*)

Calculate mean square error losses

Parameters

- **`y_true`** (*`Union(tf.Tensor, tf.Variable)`*) – Ground Truth

- **y_pred** (*Union(tf.Tensor, tf.Variable)*) – Prediction

Returns Mean Squared Error

Return type tf.Tensor

History 2017-Nov-16 - Written - Henry Leung (University of Toronto)

MSE is based on the equation

$$Loss_i = \begin{cases} (\hat{y}_i - y_i)^2 \text{ for } y_i \neq \text{Magic Number} \\ 0 \text{ for } y_i = \text{Magic Number} \end{cases}$$

And thus the loss for mini-batch is

$$Loss_{NN} = \frac{1}{D} \sum_{i=1}^{batch} (Loss_i \mathcal{F}_{correction,i})$$

It can be used with Keras, you just have to import the function from astroNN

```
def keras_model():
    # Your keras_model define here
    return model

model = keras_model()
# remember to import astroNN's loss function first
model.compile(loss=mean_squared_error, ...)
```

4.4.3 Mean Absolute Error

astroNN.nn.losses.**mean_absolute_error**(*y_true*, *y_pred*)

Calculate mean absolute error, ignoring the magic number

Parameters

- **y_true** (*Union(tf.Tensor, tf.Variable)*) – Ground Truth
- **y_pred** (*Union(tf.Tensor, tf.Variable)*) – Prediction

Returns Mean Absolute Error

Return type tf.Tensor

History 2018-Jan-14 - Written - Henry Leung (University of Toronto)

MAE is based on the equation

$$Loss_i = \begin{cases} |\hat{y}_i - y_i| \text{ for } y_i \neq \text{Magic Number} \\ 0 \text{ for } y_i = \text{Magic Number} \end{cases}$$

And thus the loss for mini-batch is

$$Loss_{NN} = \frac{1}{D} \sum_{i=1}^{batch} (Loss_i \mathcal{F}_{correction,i})$$

It can be used with Keras, you just have to import the function from astroNN

```
def keras_model():
    # Your keras_model define here
    return model

model = keras_model()
# remember to import astroNN's loss function first
model.compile(loss=mean_absolute_error, ...)
```

4.4.4 Mean Error

astroNN.nn.losses.**mean_error**(*y_true*, *y_pred*)

Calculate mean error as a way to get the bias in prediction, ignoring the magic number

Parameters

- **y_true** (*Union(tf.Tensor, tf.Variable)*) – Ground Truth
- **y_pred** (*Union(tf.Tensor, tf.Variable)*) – Prediction

Returns Mean Error

Return type tf.Tensor

History 2018-May-22 - Written - Henry Leung (University of Toronto)

Mean Error is a metrics to evaluate the bias of prediction and is based on the equation

$$Loss_i = \begin{cases} \hat{y_i} - y_i \text{ for } y_i \neq \text{Magic Number} \\ 0 \text{ for } y_i = \text{Magic Number} \end{cases}$$

And thus the loss for mini-batch is

$$Loss_{NN} = \frac{1}{D} \sum_{i=1}^{batch} (Loss_i \mathcal{F}_{correction,i})$$

It can be used with Keras, you just have to import the function from astroNN

```
def keras_model():
    # Your keras_model define here
    return model

model = keras_model()
# remember to import astroNN's loss function first
model.compile(loss=mean_error, ...)
```

4.4.5 Regression Loss and Predictive Variance Loss for Bayesian Neural Net

astroNN.nn.losses.**robust_mse**(*y_true*, *y_pred*, *variance*, *labels_err*)

Calculate predictive variance, and takes account of labels error in Bayesian Neural Network

Parameters

- **y_true** (*Union(tf.Tensor, tf.Variable)*) – Ground Truth
- **y_pred** (*Union(tf.Tensor, tf.Variable)*) – Prediction
- **variance** (*Union(tf.Tensor, tf.Variable)*) – Log Predictive Variance

- **labels_err** (*Union(tf.Tensor, tf.Variable)*) – Known labels error, give zeros if unknown/unavailable

Returns Robust Mean Squared Error, can be used directly with Tensorflow

Return type tf.Tensor

History 2018-April-07 - Written - Henry Leung (University of Toronto)

astroNN.nn.losses.**mse_lin_wrapper**(*var*, *labels_err*)

Calculate predictive variance, and takes account of labels error in Bayesian Neural Network

Parameters

- **var** (*Union(tf.Tensor, tf.Variable)*) – Predictive Variance
- **labels_err** (*Union(tf.Tensor, tf.Variable)*) – Known labels error, give zeros if unknown/unavailable

Returns Robust MSE function for labels prediction neurones, which matches Keras losses API

Return type function

Returned Funtion Parameter

function(y_true, y_pred)

- **y_true** (*tf.Tensor*): Ground Truth
- **y_pred** (*tf.Tensor*): Prediction

Return (*tf.Tensor*): Robust Mean Squared Error

History 2017-Nov-16 - Written - Henry Leung (University of Toronto)

astroNN.nn.losses.**mse_var_wrapper**(*lin*, *labels_err*)

Calculate predictive variance, and takes account of labels error in Bayesian Neural Network

Parameters

- **lin** (*Union(tf.Tensor, tf.Variable)*) – Prediction
- **labels_err** (*Union(tf.Tensor, tf.Variable)*) – Known labels error, give zeros if unknown/unavailable

Returns Robust MSE function for predictive variance neurones which matches Keras losses API

Return type function

Returned Funtion Parameter

function(y_true, y_pred)

- **y_true** (*tf.Tensor*): Ground Truth
- **y_pred** (*tf.Tensor*): Predictive Variance

Return (*tf.Tensor*): Robust Mean Squared Error

History 2017-Nov-16 - Written - Henry Leung (University of Toronto)

It is based on the equation implemented as *robust_mse()*, please notice s_i is representing $log((\sigma_{predictive,i})^2 + (\sigma_{known,i})^2)$. Neural network not predicting variance directly to avoid numerical instability but predicting $log((\sigma_i)^2)$

$$Loss_i = \begin{cases} \frac{1}{2}(\hat{y_i} - y_i)^2 e^{-s_i} + \frac{1}{2}(s_i) \text{ for } y_i \neq \text{Magic Number} \\ 0 \text{ for } y_i = \text{Magic Number} \end{cases}$$

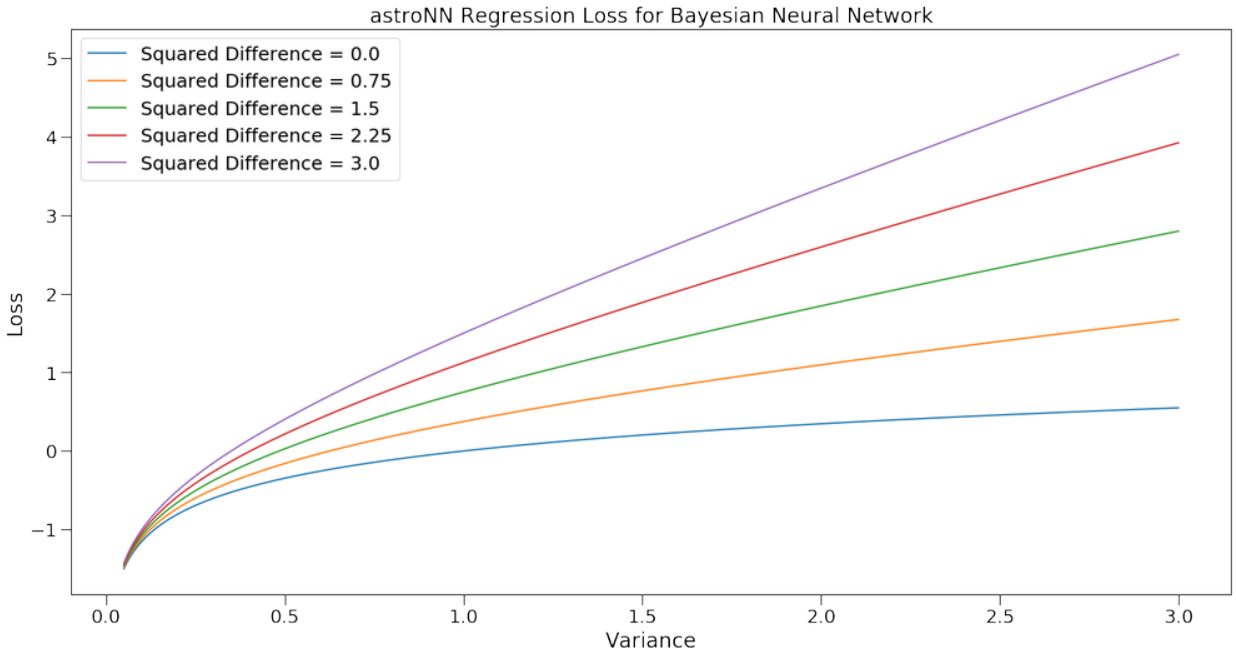
And thus the loss for mini-batch is

$$Loss_{BNN} = \frac{1}{D} \sum_{i=1}^{batch} (Loss_i \mathcal{F}_{correction,i})$$

They basically do the same things and can be used with Keras, you just have to import the functions from astroNN

```
def keras_model():
    # Your keras_model define here

    # model for the training process
    model = Model(inputs=[input_tensor, labels_err_tensor], outputs=[output, variance_
↪output])

    # model for the prediction
    model_prediction = Model(inputs=input_tensor, outputs=[output, variance_output])

    variance_output = Dense(name='variance_output', ...)
    output = Dense(name='output', ...)

    predictive_variance_loss = mse_var_wrapper(output, labels_err_tensor)
    output_loss = mse_lin_wrapper(predictive_variance, labels_err_tensor)

    return model, model_prediction, output_loss, predictive_variance_loss

model, model_prediction, output_loss, predictive_variance_loss = keras_model()
# remember to import astroNN loss function first
model.compile(loss={'output': output_loss, 'variance_output': predictive_variance_
↪loss}, ...)
```

To better understand this loss function, you can see the following plot of Loss vs Variance colored by squared difference which is $(\hat{y}_i - y_i)^2$



4.4.6 Mean Squared Logarithmic Error

`astroNN.nn.losses.`**`mean_squared_logarithmic_error`**(*y_true*, *y_pred*)

Calculate mean squared logarithmic error, ignoring the magic number

Parameters

- **y_true** (*Union(tf.Tensor, tf.Variable)*) – Ground Truth
- **y_pred** (*Union(tf.Tensor, tf.Variable)*) – Prediction

Returns Mean Squared Logarithmic Error

Return type tf.Tensor

History 2018-Feb-17 - Written - Henry Leung (University of Toronto)

MSLE will first clip the values of prediction from neural net for the sake of numerical stability,

$$y_i = \begin{cases} \epsilon + 1 \text{ for } y_i < \epsilon \\ y_i + 1 \text{ for otherwise} \end{cases}$$

where ϵ is a small constant

Then MSLE is based on the equation

$$Loss_i = \begin{cases} (\log(\hat{y_i}) - \log(y_i))^2 \text{ for } y_i \neq \text{Magic Number} \\ 0 \text{ for } y_i = \text{Magic Number} \end{cases}$$

And thus the loss for mini-batch is

$$Loss_{NN} = \frac{1}{D} \sum_{i=1}^{batch} (Loss_i \mathcal{F}_{correction,i})$$

It can be used with Keras, you just have to import the function from astroNN

```
def keras_model():
    # Your keras_model define here
    return model

model = keras_model()
# remember to import astroNN's loss function first
model.compile(loss=mean_squared_logarithmic_error, ...)
```

4.4.7 Mean Absolute Percentage Error

astroNN.nn.losses.**mean_absolute_percentage_error**(*y_true*, *y_pred*)

Calculate mean absolute percentage error, ignoring the magic number

Parameters

- **y_true** (*Union(tf.Tensor, tf.Variable)*) – Ground Truth
- **y_pred** (*Union(tf.Tensor, tf.Variable)*) – Prediction

Returns Mean Absolute Percentage Error

Return type tf.Tensor

History 2018-Feb-17 - Written - Henry Leung (University of Toronto)

Mean Absolute Percentage Error will first clip the values of prediction from neural net for the sake of numerical stability,

$$y_i = \begin{cases} \epsilon \text{ for } y_i < \epsilon \\ y_i \text{ for otherwise} \end{cases}$$

where ϵ is a small constant

Then Mean Absolute Percentage Error is based on the equation

$$Loss_i = \begin{cases} 100 \dfrac{|y_i - \hat{y_i}|}{y_i} \text{ for } y_i \neq \text{Magic Number} \\ 0 \text{ for } y_i = \text{Magic Number} \end{cases}$$

And thus the loss for mini-batch is

$$Loss_{NN} = \frac{1}{D} \sum_{i=1}^{batch} (Loss_i \mathcal{F}_{correction,i})$$

It can be used with Keras, you just have to import the function from astroNN

```
def keras_model():
    # Your keras_model define here
    return model

model = keras_model()
# remember to import astroNN's loss function first
model.compile(loss=mean_absolute_percentage_error, ...)
```

4.4.8 Mean Percentage Error

astroNN.nn.losses.**mean_percentage_error**(*y_true*, *y_pred*)

Calculate mean percentage error, ignoring the magic number

Parameters

- **y_true** (*Union(tf.Tensor, tf.Variable)*) – Ground Truth
- **y_pred** (*Union(tf.Tensor, tf.Variable)*) – Prediction

Returns Mean Percentage Error

Return type tf.Tensor

History 2018-Jun-06 - Written - Henry Leung (University of Toronto)

Mean Percentage Error will first clip the values of prediction from neural net for the sake of numerical stability,

$$y_i = \begin{cases} \epsilon \text{ for } y_i < \epsilon \\ y_i \text{ for otherwise} \end{cases}$$

where ϵ is a small constant

Then Mean Percentage Error is based on the equation

$$Loss_i = \begin{cases} 100 \dfrac{y_i - \hat{y_i}}{y_i} \text{ for } y_i \neq \text{Magic Number} \\ 0 \text{ for } y_i = \text{Magic Number} \end{cases}$$

And thus the loss for mini-batch is

$$Loss_{NN} = \frac{1}{D} \sum_{i=1}^{batch} (Loss_i \mathcal{F}_{correction,i})$$

It can be used with Keras, you just have to import the function from astroNN

```
def keras_model():
    # Your keras_model define here
    return model

model = keras_model()
# remember to import astroNN's loss function first
model.compile(loss=mean_percentage_error, ...)
```

4.4.9 Categorical Cross-Entropy

astroNN.nn.losses.**categorical_crossentropy**(*y_true*, *y_pred*, *from_logits=False*)

Categorical cross-entropy between an output tensor and a target tensor, ignoring the magic number

Parameters

- **y_true** (*Union(tf.Tensor, tf.Variable)*) – Ground Truth
- **y_pred** (*Union(tf.Tensor, tf.Variable)*) – Prediction
- **from_logits** (*boolean*) – From logits space or not. If you want to use logits, please use from_logits=True

Returns Categorical Cross-Entropy

Return type tf.Tensor

History 2018-Jan-14 - Written - Henry Leung (University of Toronto)

Categorical Cross-Entropy will first clip the values of prediction from neural net for the sake of numerical stability if the prediction is not coming from logits (before softmax activated)

$$\hat{y}_i = \begin{cases} \epsilon \text{ for } \hat{y}_i < \epsilon \\ 1 - \epsilon \text{ for } \hat{y}_i > 1 - \epsilon \\ \hat{y}_i \text{ for otherwise} \end{cases}$$

where ϵ is a small constant

and then based on the equation

$$Loss_i = \begin{cases} y_i \log(\hat{y}_i) \text{ for } y_i \neq \text{Magic Number} \\ 0 \text{ for } y_i = \text{Magic Number} \end{cases}$$

And thus the loss for mini-batch is

$$Loss_{NN} = -\frac{1}{D} \sum_{i=1}^{batch} (Loss_i \mathcal{F}_{correction,i})$$

It can be used with Keras, you just have to import the function from astroNN

```
def keras_model():
    # Your keras_model define here
    return model

model = keras_model()
# remember to import astroNN's loss function first
model.compile(loss=categorical_crossentropy(from_logits=False), ...)
```

4.4.10 Binary Cross-Entropy

astroNN.nn.losses.**binary_crossentropy**(*y_true*, *y_pred*, *from_logits=False*)

Binary cross-entropy between an output tensor and a target tensor, ignoring the magic number

Parameters

- **y_true** (*Union(tf.Tensor, tf.Variable)*) – Ground Truth
- **y_pred** (*Union(tf.Tensor, tf.Variable)*) – Prediction
- **from_logits** (*boolean*) – From logits space or not. If you want to use logits, please use from_logits=True

Returns Binary Cross-Entropy

Return type tf.Tensor

History 2018-Jan-14 - Written - Henry Leung (University of Toronto)

Binary Cross-Entropy will first clip the values of prediction from neural net for the sake of numerical stability if `from_logits=False`

$$\hat{y}_i = \begin{cases} \epsilon \text{ for } \hat{y}_i < \epsilon \\ 1-\epsilon \text{ for } \hat{y}_i > 1-\epsilon \\ \hat{y}_i \text{ for otherwise} \end{cases}$$

where ϵ is a small constant

and then based on the equation

$$Loss_i = \begin{cases} y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i) \text{ for } y_i \neq \text{Magic Number} \\ \hat{y}_i \log(\hat{y}_i) + (1-\hat{y}_i)\log(1-\hat{y}_i) \text{ for } y_i = \text{Magic Number} \end{cases}$$

to avoid numerical instability if `from_logits=True`, we can reformulate it as

$$Loss_i = \begin{cases} \max(\hat{y}_i, 0) - y_i\hat{y}_i + \log(1+e^{-\|\hat{y}_i\|}) \text{ for } y_i \neq \text{Magic Number} \\ 0 \text{ for } y_i = \text{Magic Number} \end{cases}$$

And thus the loss for mini-batch is

$$Loss_{NN} = -\frac{1}{D}\sum_{i=1}^{batch}(Loss_i \mathcal{F}_{correction,i})$$

It can be used with Keras, you just have to import the function from astroNN

```
def keras_model():
    # Your keras_model define here
    return model

model = keras_model()
# remember to import astroNN's loss function first
model.compile(loss=binary_crossentropy(from_logits=False), ...)
```

4.4.11 Categorical Cross-Entropy and Predictive Logits Variance for Bayesian Neural Net

astroNN.nn.losses.**robust_categorical_crossentropy**(*y_true*, *y_pred*, *logit_var*)

Calculate categorical accuracy, ignoring the magic number

Parameters

- **y_true** (*Union(tf.Tensor, tf.Variable)*) – Ground Truth
- **y_pred** (*Union(tf.Tensor, tf.Variable)*) – Prediction in logits space
- **logit_var** (*Union(tf.Tensor, tf.Variable)*) – Predictive variance in logits space

Returns categorical cross-entropy

Return type tf.Tensor

History 2018-Mar-15 - Written - Henry Leung (University of Toronto)

astroNN.nn.losses.**bayesian_categorical_crossentropy_wrapper**(*logit_var*)

Categorical crossentropy between an output tensor and a target tensor for Bayesian Neural Network equation (12) of arxiv:1703.04977

Parameters **logit_var** (*Union(tf.Tensor, tf.Variable)*) – Predictive variance

Returns Robust categorical_crossentropy function for predictive variance neurones which matches Keras losses API

Return type function

Returned Function Parameter

function(y_true, y_pred)
- **y_true** (*tf.Tensor*): Ground Truth
- **y_pred** (*tf.Tensor*): Prediction in logits space

Return (*tf.Tensor*): Robust categorical crossentropy

History 2018-Mar-15 - Written - Henry Leung (University of Toronto)

astroNN.nn.losses.**bayesian_categorical_crossentropy_var_wrapper**(*logits*)

Categorical crossentropy between an output tensor and a target tensor for Bayesian Neural Network equation (12) of arxiv:1703.04977

Parameters **logits** (*Union(tf.Tensor, tf.Variable)*) – Prediction in logits space

Returns Robust categorical_crossentropy function for predictive variance neurones which matches Keras losses API

Return type function

Returned Function Parameter

function(y_true, y_pred)
- **y_true** (*tf.Tensor*): Ground Truth
- **y_pred** (*tf.Tensor*): Predictive variance in logits space

Return (*tf.Tensor*): Robust categorical crossentropy

History 2018-Mar-15 - Written - Henry Leung (University of Toronto)

It is based on Equation 12 from arxiv:1703.04977. s_i is representing the predictive variance of logits

$$Loss_i = \begin{cases} \text{Categorical Cross-Entropy} + \text{Distorted Categorical Cross-Entropy} + e^{s_i} - 1 \text{ for } y_i \neq \text{Magic Number} \\ 0 \text{ for } y_i = \text{Magic Number} \end{cases}$$

where *Distorted Categorical Cross-Entropy* is defined as

$$\text{elu}(\text{Categorical Cross-Entropy}(y_i, \hat{y}_i) - \text{Categorical Cross-Entropy}(y_i, \mathcal{N}(\hat{y}_i, \sqrt{s_i})))$$

And thus the loss for mini-batch is

$$Loss_{BNN} = \frac{1}{D}\sum_{i=1}^{batch}(Loss_i \mathcal{F}_{correction,i})$$

bayesian_categorical_crossentropy_wrapper is for the prediction neurones

bayesian_categorical_crossentropy_var_wrapper is for the predictive variance neurones

They basically do the same things and can be used with Keras, you just have to import the functions from astroNN

```
def keras_model():
    # Your keras_model define here

    # model for the training process
    model = Model(inputs=[input_tensor], outputs=[output, variance_output])

    # model for the prediction
    model_prediction = Model(inputs=input_tensor, outputs=[output, variance_output])

    variance_output = Dense(name='predictive_variance', ...)
    output = Dense(name='output', ...)

    predictive_variance_loss = bayesian_categorical_crossentropy_var_wrapper(output)
    output_loss = bayesian_categorical_crossentropy_wrapper(predictive_variance)

    return model, model_prediction, output_loss, predictive_variance_loss

model, model_prediction, output_loss, predictive_variance_loss = keras_model()
# remember to import astroNN loss function first
model.compile(loss={'output': output_loss, 'variance_output': predictive_variance_
↪loss}, ...)
```

4.4.12 Binary Cross-Entropy and Predictive Logits Variance for Bayesian Neural Net

astroNN.nn.losses.**robust_binary_crossentropy**(*y_true*, *y_pred*, *logit_var*)

Calculate binary accuracy, ignoring the magic number

Parameters

- **y_true** (*Union(tf.Tensor, tf.Variable)*) – Ground Truth
- **y_pred** (*Union(tf.Tensor, tf.Variable)*) – Prediction in logits space
- **logit_var** (*Union(tf.Tensor, tf.Variable)*) – Predictive variance in logits space

Returns categorical cross-entropy

Return type tf.Tensor

History 2018-Mar-15 - Written - Henry Leung (University of Toronto)

astroNN.nn.losses.**bayesian_binary_crossentropy_wrapper**(*logit_var*)

Binary crossentropy between an output tensor and a target tensor for Bayesian Neural Network
equation (12) of arxiv:1703.04977

Parameters **logit_var** (*Union(tf.Tensor, tf.Variable)*) – Predictive variance

Returns Robust binary_crossentropy function for predictive variance neurones which matches Keras losses API

Return type function

Returned Function Parameter

function(y_true, y_pred)
- **y_true** (*tf.Tensor*): Ground Truth
- **y_pred** (*tf.Tensor*): Prediction in logits space

Return (*tf.Tensor*): Robust binary crossentropy

History 2018-Mar-15 - Written - Henry Leung (University of Toronto)

astroNN.nn.losses.**bayesian_binary_crossentropy_var_wrapper**(*logits*)

Binary crossentropy between an output tensor and a target tensor for Bayesian Neural Network
equation (12) of arxiv:1703.04977

Parameters **logits** (*Union(tf.Tensor, tf.Variable)*) – Prediction in logits space

Returns Robust binary_crossentropy function for predictive variance neurones which matches Keras losses API

Return type function

Returned Function Parameter

function(y_true, y_pred)
- **y_true** (*tf.Tensor*): Ground Truth
- **y_pred** (*tf.Tensor*): Predictive variance in logits space

Return (*tf.Tensor*): Robust binary crossentropy

History 2018-Mar-15 - Written - Henry Leung (University of Toronto)

It is based on Equation 12 from arxiv:1703.04977. s_i is representing the predictive variance of logits

$$Loss_i = \begin{cases} \text{Binary Cross-Entropy} + \text{Distorted Binary Cross-Entropy} + e^{s_i} - 1 & \text{for } y_i \neq \text{Magic Number} \\ 0 & \text{for } y_i = \text{Magic Number} \end{cases}$$

where *Distorted Binary Cross-Entropy* is defined as

$$\text{elu}(\text{Binary Cross-Entropy}(y_i, \hat{y}_i) - \text{Binary Cross-Entropy}(y_i, \mathcal{N}(\hat{y}_i, \sqrt{s_i})))$$

And thus the loss for mini-batch is

$$Loss_{BNN} = \frac{1}{D} \sum_{i=1}^{batch} (Loss_i \mathcal{F}_{correction,i})$$

bayesian_binary_crossentropy_wrapper is for the prediction neurones

bayesian_binary_crossentropy_var_wrapper is for the predictive variance neurones

They basically do the same things and can be used with Keras, you just have to import the functions from astroNN

```
def keras_model():
    # Your keras_model define here

    # model for the training process
    model = Model(inputs=[input_tensor], outputs=[output, variance_output])

    # model for the prediction
    model_prediction = Model(inputs=input_tensor, outputs=[output, variance_output])

    variance_output = Dense(name='predictive_variance', ...)
    output = Dense(name='output', ...)

    predictive_variance_loss = bayesian_binary_crossentropy_var_wrapper(output)
    output_loss = bayesian_binary_crossentropy_wrapper(predictive_variance)

    return model, model_prediction, output_loss, predictive_variance_loss

model, model_prediction, output_loss, predictive_variance_loss = keras_model()
# remember to import astroNN loss function first
model.compile(loss={'output': output_loss, 'variance_output': predictive_variance_
↪loss}, ...)
```

4.4.13 Categorical Classification Accuracy

astroNN.nn.losses.**categorical_accuracy**(*y_true*, *y_pred*)

Calculate categorical accuracy, ignoring the magic number

Parameters

- **y_true** (*Union(tf.Tensor, tf.Variable)*) – Ground Truth
- **y_pred** (*Union(tf.Tensor, tf.Variable)*) – Prediction

Returns Categorical Classification Accuracy

Return type tf.Tensor

History 2018-Jan-21 - Written - Henry Leung (University of Toronto)

Categorical Classification Accuracy will first deal with Magic Number

$$Loss_i = \begin{cases} y_i \text{ for } y_i \neq \text{Magic Number} \\ 0 \text{ for } y_i = \text{Magic Number} \end{cases}$$

Then based on the equation

$$Accuracy_i = \begin{cases} 1 \text{ for Argmax}(y_i) = \text{Argmax}(\hat{y_i}) \\ 0 \text{ for Argmax}(y_i) \neq \text{Argmax}(\hat{y_i}) \end{cases}$$

And thus the accuracy for is

$$Accuracy = \frac{1}{D} \sum_{i=1}^{labels} (Accuracy_i \mathcal{F}_{correction,i})$$

It can be used with Keras, you just have to import the function from astroNN

```
def keras_model():
    # Your keras_model define here
    return model

model = keras_model()
# remember to import astroNN's metrics function first
model.compile(metrics=categorical_accuracy, ...)
```

Note: Please make sure you use categorical_accuracy when using categorical_crossentropy as the loss function

4.4.14 Binary Classification Accuracy

astroNN.nn.losses.**binary_accuracy**(*from_logits=False*)

Calculate binary accuracy, ignoring the magic number

Parameters **from_logits** (*boolean*) – From logits space or not. If you want to use logits, please use from_logits=True

Returns Function for Binary classification accuracy which matches Keras losses API

Return type function

Returned Funtion Parameter

function(y_true, y_pred)

- **y_true** (*tf.Tensor*): Ground Truth
- **y_pred** (*tf.Tensor*): Prediction

Return (*tf.Tensor*): Binary Classification Accuracy

History 2018-Jan-31 - Written - Henry Leung (University of Toronto)

Binary Classification Accuracy will round the values of prediction if `from_logits=False` or will apply sigmoid first and then round the values of prediction if `from_logits=True`

$$\hat{y_i} = \begin{cases} 1 \text{ for } \hat{y_i} > 0.5 \\ 0 \text{ for } \hat{y_i} \leq 0.5 \end{cases}$$

and then based on the equation

$$Accuracy_i = \begin{cases} 1 \text{ for } y_i = \hat{y_i} \\ 0 \text{ for } y_i \neq \hat{y_i} \end{cases}$$

And thus the accuracy for is

$$Accuracy = \frac{1}{D} \sum_{i=1}^{labels} (Accuracy_i \mathcal{F}_{correction,i})$$

It can be used with Keras, you just have to import the function from astroNN

```
def keras_model():
    # Your keras_model define here
    return model

model = keras_model()
# remember to import astroNN's metrics function first
model.compile(metrics=binary_accuracy(from_logits=False), ...)
```

Note: Please make sure you use binary_accuracy when using binary_crossentropy as the loss function

4.4.15 Zeros Loss

astroNN.nn.losses.**zeros_loss**(*y_true*, *y_pred*)

Always return zeros

Parameters

- **y_true** (*Union(tf.Tensor, tf.Variable)*) – Ground Truth
- **y_pred** (*Union(tf.Tensor, tf.Variable)*) – Prediction

Returns Zeros

Return type tf.Tensor

History 2018-May-24 - Written - Henry Leung (University of Toronto)

zeros_loss is a loss function that will always return zero loss and the function matches Keras API. It is mainly designed to do testing or experiments.

It can be used with Keras, you just have to import the function from astroNN

```
def keras_model():
    # Your keras_model define here
    return model

model = keras_model()
# remember to import astroNN's loss function first
model.compile(loss=zeros_loss, ...)
```

4.5 Layers - astroNN.nn.layers

astroNN provides some customized layers which built on Keras and Tensorflow. Thus they are compatible with Keras with Tensorflow backend or tensorflow.keras. You can just treat astroNN customized layers as conventional Keras layers.

4.5.1 Monte Carlo Dropout Layer

class astroNN.nn.layers.**MCDropout**(*rate*, *disable=False*, *noise_shape=None*, *name=None*, ***kwargs*)

Dropout Layer for Bayesian Neural Network, this layer will always on regardless the learning phase flag

Parameters

- **rate** (*float*) – Dropout Rate between 0 and 1
- **disable** (*boolean*) – Dropout on or off

Returns A layer

Return type object

History 2018-Feb-05 - Written - Henry Leung (University of Toronto)

call(*inputs*, *training=None*)

Note Equivalent to __call__()

Parameters **inputs** (*tf.Tensor*) – Tensor to be applied

Returns Tensor after applying the layer

Return type tf.Tensor

get_config()

Returns Dictionary of configuration

Return type dict

MCDropout is basically Keras's Dropout layer without *seed* argument support. Moreover, the layer will ignore Keras's learning phase flag, so the layer will always stays on even in prediction phase.

Dropout can be described by the following formula, lets say we have i neurones after activation with value y_i

$$r_i = \text{Bernoulli}(p)$$
$$\hat{y_i} = r_i * y_i$$

It can be used with keras or tensorflow.keras, you just have to import the function from astroNN

```
def keras_model():
    # Your keras_model define here, assuming you are using functional API
    b_dropout = MCDropout(0.2)(some_keras_layer)
    return model
```

If you really want to disable the dropout, you do it by

```
# Your keras_model define here, assuming you are using functional API
b_dropout = MCDropout(0.2, disable=True)(some_keras_layer)
```

4.5.2 Monte Carlo Dropout with Continuous Relaxation Layer Wrapper

class astroNN.nn.layers.**MCConcreteDropout**(*layer*, *weight_regularizer=5e-13*, *dropout_regularizer=0.0001*, *init_min=0.1*, *init_max=0.2*, *disable=False*, ***kwargs*)

Monte Carlo Dropout with Continuous Relaxation Layer Wrapper This layer will learn the dropout probability arXiv:1705.07832

Parameters **layer** (*keras.layers.Layer*) – The layer to be applied concrete dropout

Returns A layer

Return type object

History 2018-Mar-04 - Written - Henry Leung (University of Toronto)

call(*inputs*, *training=None*)

Note Equivalent to __call__()

Parameters **inputs** (*tf.Tensor*) – Tensor to be applied

Returns Tensor after applying the layer

Return type tf.Tensor

get_config()

Returns Dictionary of configuration

Return type dict

MCConcreteDropout is an implementation of arXiv:1705.07832, modified from the original implementation here. Moreover, the layer will ignore Keras's learning phase flag, so the layer will always stays on even in prediction phase. This layer should be only used for experimental purpose only as it has not been tested rigorously. *MCConcreteDropout* is technically a layer wrapper instead of a standard layer, so it needs to take a layer as an input argument.

The main difference between *MCConcreteDropout* and standard bernoulli dropout is *MCConcreteDropout* learns dropout rate during training instead of a fixed probability. Turning/learning dropout rate is not a novel idea, it can be traced back to one of the original paper arXiv:1506.02557 on variational dropout. But *MCConcreteDropout* focuses on the role and importance of dropout with Bayesian technique.

It can be used with keras or tensorflow.keras, you just have to import the function from astroNN

```
def keras_model():
    # Your keras_model define here, assuming you are using functional API
    c_dropout = MCConcreteDropout(some_keras_layer)(previous_layer)
    return model
```

If you really want to disable the dropout, you do it by

```
# Your keras_model define here, assuming you are using functional API
c_dropout = MCConcreteDropout((some_keras_layer), disable=True)(previous_layer)
```

4.5.3 Monte Carlo Spatial Dropout Layer

MCSpatialDropout1D should be used with Conv1D and *MCSpatialDropout2D* should be used with Conv2D

class astroNN.nn.layers.**MCSpatialDropout1D**(*rate*, *disable=False*, ***kwargs*)

Spatial 1D version of Dropout of Dropout Layer for Bayesian Neural Network, this layer will always regardless the learning phase flag

Parameters

- **rate** (*float*) – Dropout Rate between 0 and 1
- **disable** (*boolean*) – Dropout on or off

Returns A layer

Return type object

History 2018-Mar-07 - Written - Henry Leung (University of Toronto)

call(*inputs*, *training=None*)

Note Equivalent to __call__()

Parameters **inputs** (*tf.Tensor*) – Tensor to be applied

Returns Tensor after applying the layer

Return type tf.Tensor

get_config()

Returns Dictionary of configuration

Return type dict

class astroNN.nn.layers.**MCSpatialDropout2D**(*rate*, *disable=False*, ***kwargs*)

Spatial 2D version of Dropout of Dropout Layer for Bayesian Neural Network, this layer will always regardless the learning phase flag

Parameters

- **rate** (*float*) – Dropout Rate between 0 and 1
- **disable** (*boolean*) – Dropout on or off

Returns A layer

Return type object

History 2018-Mar-07 - Written - Henry Leung (University of Toronto)

call(*inputs*, *training=None*)

Note Equivalent to __call__()

Parameters **inputs** (*tf.Tensor*) – Tensor to be applied

Returns Tensor after applying the layer

Return type tf.Tensor

get_config()

Returns Dictionary of configuration

Return type dict

MCSpatialDropout1D and *MCSpatialDropout2D* are basically Keras's Spatial Dropout layer without *seed* and *noise_shape* argument support. Moreover, the layers will ignore Keras's learning phase flag, so the layers will always stays on even in prediction phase.

This version performs the same function as Dropout, however it drops entire 1D feature maps instead of individual elements. If adjacent frames within feature maps are strongly correlated (as is normally the case in early convolution layers) then regular dropout will not regularize the activations and will otherwise just result in an effective learning rate decrease. In this case, SpatialDropout1D will help promote independence between feature maps and should be used instead.

For technical detail, you can refer to the original paper arXiv:1411.4280

It can be used with keras or tensorflow.keras, you just have to import the function from astroNN

```
def keras_model():
    # Your keras_model define here, assuming you are using functional API
    b_dropout = MCSpatialDropout1D(0.2)(keras_conv_layer)
    return model
```

If you really want to disable the dropout, you do it by

```
# Your keras_model define here, assuming you are using functional API
b_dropout = MCSpatialDropout1D(0.2, disable=True)(keras_conv_layer)
```

4.5.4 Monte Carlo Gaussian Dropout Layer

class astroNN.nn.layers.**MCGaussianDropout**(*rate*, *disable=False*, *name=None*, ***kwargs*)

Dropout Layer for Bayesian Neural Network, this layer will always on regardless the learning phase flag standard deviation sqrt(rate / (1 - rate))

Parameters

- **rate** (*float*) – Dropout Rate between 0 and 1
- **disable** (*boolean*) – Dropout on or off

Returns A layer

Return type object

History 2018-Mar-07 - Written - Henry Leung (University of Toronto)

call(*inputs*, *training=None*)

Note Equivalent to __call__()

Parameters **inputs** (*tf.Tensor*) – Tensor to be applied

Returns Tensor after applying the layer

Return type tf.Tensor

get_config()

Returns Dictionary of configuration

Return type dict

MCGaussianDropout is basically Keras's Dropout layer without *seed* argument support. Moreover, the layer will ignore Keras's learning phase flag, so the layer will always stays on even in prediction phase.

MCGaussianDropout should be used with caution for Bayesian Neural Network: https://arxiv.org/abs/1711.02989

Gaussian Dropout can be described by the following formula, lets say we have i neurones after activation with value y_i

$$r_i = \mathcal{N}\left(1, \sqrt{\frac{p}{1-p}}\right)$$
$$\hat{y}_i = r_i * y_i$$

It can be used with keras or tensorflow.keras, you just have to import the function from astroNN

```
def keras_model():
    # Your keras_model define here, assuming you are using functional API
    b_dropout = MCGaussianDropout(0.2)(some_keras_layer)
    return model
```

If you really want to disable the dropout, you do it by

```
# Your keras_model define here, assuming you are using functional API
b_dropout = MCGaussianDropout(0.2, disable=True)(some_keras_layer)
```

4.5.5 Monte Carlo Batch Normalization Layer

class astroNN.nn.layers.**MCBatchNorm**(*disable=False*, *name=None*, ***kwargs*)

Monte Carlo Batch Normalization Layer for Bayesian Neural Network

Parameters **disable** (*boolean*) – Dropout on or off

Returns A layer

Return type object

History 2018-Apr-12 - Written - Henry Leung (University of Toronto)

call(*inputs*, *training=None*)

Note Equivalent to __call__()

Parameters **inputs** (*tf.Tensor*) – Tensor to be applied

Returns Tensor after applying the layer

Return type tf.Tensor

get_config()

Returns Dictionary of configuration

Return type dict

MCBatchNorm is a layer doing Batch Normalization originally described in arViX: https://arxiv.org/abs/1502.03167

MCBatchNorm should be used with caution for Bayesian Neural Network: https://openreview.net/forum?id=BJlrSmbAZ

Batch Normalization can be described by the following formula, lets say we have N neurones after activation for a layer

$$N_i = \frac{N_i - \text{Mean}[N]}{\sqrt{\text{Var}[N]}}$$

MCBatchNorm can be imported by

```
from astroNN.nn.layers import MCBatchNorm
```

It can be used with keras or tensorflow.keras, you just have to import the function from astroNN

```
def keras_model():
    # Your keras_model define here, assuming you are using functional API
    b_dropout = MCBatchNorm()(some_keras_layer)
    return model
```

4.5.6 Error Propagation Layer

class astroNN.nn.layers.**ErrorProp**(*stddev*, *name=None*, ***kwargs*)

Propagate Error Layer, do nothing during training, add gaussian noise during testing phase

Parameters **stddev** (*float*) – Known 1-S.D. Uncertainty in input data

Returns A layer

Return type object

History 2018-Feb-05 - Written - Henry Leung (University of Toronto)

call(*inputs*, *training=None*)

Note Equivalent to __call__()

Parameters **inputs** (*tf.Tensor*) – Tensor to be applied

Returns Tensor after applying the layer

Return type tf.Tensor

get_config()

Returns Dictionary of configuration

Return type dict

ErrorProp is a layer designed to do error propagation in neural network. It will acts as an identity transformation layer during training phase but add gaussian noise to input during test phase. The idea is if you have known uncertainty in input, and you want to understand how input uncertainty (more specifically this layer assuming the uncertainty is Gaussian) affects the output. Since this layer add random known Gaussian uncertainty to the input, you can run model prediction a few times to get some predictions, mean of those predictions will be the final prediction and standard derivation of the predictions will be the propagated uncertainty.

ErrorProp can be imported by

```
from astroNN.nn.layers import ErrorProp
```

It can be used with keras or tensorflow.keras, you just have to import the function from astroNN

```
def keras_model():
    # Your keras_model define here, assuming you are using functional API
    input = Input(.....)
    input_with_error = ErrorProp(some_gaussian_tensor)(input)
    return model
```

4.5.7 KL-Divergence Layer for Variational Autoencoder

class astroNN.nn.layers.**KLDivergenceLayer**(*name=None*, ***kwargs*)

Identity transform layer that adds KL divergence to the final model losses.
KL divergence used to force the latent space match the prior (in this case its unit gaussian)

Returns A layer

Return type object

History 2018-Feb-05 - Written - Henry Leung (University of Toronto)

call(*inputs*, *training=None*)

Note Equivalent to __call__()

Parameters **inputs** (*tf.Tensor*) – Tensor to be applied, concatenated tf.tensor of mean and std in latent space

Returns Tensor after applying the layer

Return type tf.Tensor

get_config()

Returns Dictionary of configuration

Return type dict

KLDivergenceLayer is a layer designed to be used in Variational Autoencoder. It will acts as an identity transformation layer but will add KL-divergence to the total loss.

KLDivergenceLayer can be imported by

```
from astroNN.nn.layers import KLDivergenceLayer
```

It can be used with keras or tensorflow.keras, you just have to import the function from astroNN

```
def keras_model():
    # Your keras_model define here, assuming you are using functional API
    z_mu = Encoder_Mean_Layer(.....)
    z_log_var = Encoder_Var_Layer(.....)
    z_mu, z_log_var = KLDivergenceLayer()([z_mu, z_log_var])
    # And then decoder or whatever
    return model
```

4.5.8 Polynomial Fitting Layer

class astroNN.nn.layers.**PolyFit**(*deg=1*, *output_units=1*, *use_xbias=True*, *init_w=None*, *name=None*, *activation=None*, *kernel_regularizer=None*, *kernel_constraint=None*)

n-deg polynomial fitting layer which acts as an neural network layer to be optimized

Parameters

- **deg** (*int*) – degree of polynomial
- **output_units** (*int*) – number of output neurons
- **use_xbias** (*bool*) – If True, then fitting output=P(inputs)+inputs, else fitting output=P(inputs)
- **init_w** (*Union[NoneType, list]*) – [Optional] list of initial weights if there is any, the list should be [n-degree, input_size, output_size]
- **name** (*Union[NoneType, str]*) – [Optional] name of the layer
- **activation** (*Union[NoneType, str]*) – [Optional] activation, default is 'linear'
- **kernel_regularizer** (*Union[NoneType, str]*) – [Optional] kernel regularizer
- **kernel_constraint** (*Union[NoneType, str]*) – [Optional] kernel constraint

Returns A layer

Return type object

History 2018-Jul-24 - Written - Henry Leung (University of Toronto)

call(*inputs*)

Note Equivalent to __call__()

Parameters **inputs** (*tf.Tensor*) – Tensor to be applied

Returns Tensor after applying the layer which is just n-deg P(inputs)

Return type tf.Tensor

get_config()

Returns Dictionary of configuration

Return type dict

PolyFit is a layer designed to do n-degree polynomial fitting in a neural network style by treating coefficient as neural network weights and optimize them by neural network optimizer. The fitted polynomial(s) are in the following form (you can specify initial weights by init_w=[[[w_0]], [[w_1]], ..., [[w_n]]]) for a single input and output value

$$p(x) = w_0 + w_1 * x + ... + w_n * x^n$$

For multiple i input values and j output values and n-deg polynomial (you can specify initial weights by init_w=[[[$w_{0,1,0}$, $w_{0,1,1}$, ..., $w_{0,1,j}$], [$w_{0,2,0}$, $w_{0,2,1}$, ..., $w_{0,2,j}$], ... [$w_{0,i,0}$, $w_{0,i,1}$, ..., $w_{0,i,j}$]], ..., [[$w_{n,1,0}$, $w_{n,1,1}$, ..., $w_{n,1,j}$], [$w_{n,2,0}$, $w_{n,2,1}$, ..., $w_{n,2,j}$], ... [$w_{n,i,0}$, $w_{n,i,1}$, ..., $w_{n,i,j}$]]])

and the polynomial is as the following form for For multiple i input values and j output values and n-deg polynomial

$$\text{output neurons from 1 to j} = \begin{cases} p_1(x) = \sum_{i=1}^{i} \left(w_{0,1,0} + w_{1,1,1} * x_1 + ... + w_{n,1,i} * x_i^n \right) \\ p_2(x) = \sum_{i=1}^{i} \left(w_{0,2,0} + w_{1,2,1} * x_1 + ... + w_{n,2,i} * x_i^n \right) \\ p_{...}(x) = \sum_{i=1}^{i} \left(..... \right) \\ p_j(x) = \sum_{i=1}^{i} \left(w_{0,j,0} + w_{1,j,1} * x_1 + ... + w_{n,j,i} * x_i^n \right) \end{cases}$$

PolyFit can be imported by

```
from astroNN.nn.layers import PolyFit
```

It can be used with keras or tensorflow.keras, you just have to import the function from astroNN

```
def keras_model():
    # Your keras_model define here, assuming you are using functional API
    input = Input(.....)
    output = PolyFit(deg=1)(input)
    return model(inputs=input, outputs=output)
```

To show it works as a polynomial, you can refer the following example:

```
import numpy as np
from astroNN.nn.layers import PolyFit

from astroNN.shared.nn_tools import cpu_fallback
from astroNN.config import keras_import_manager

keras = keras_import_manager()  # either import keras or tf.keras
cpu_fallback()  # force tf to use CPU

Input = keras.layers.Input
Model = keras.models.Model

# Data preparation
polynomial_coefficient = [0.1, -0.05]
random_xdata = np.random.normal(0, 3, (100, 1))
random_ydata = polynomial_coefficient[1] * random_xdata + polynomial_coefficient[0]

input = Input(shape=[1, ])
# set initial weights
output = PolyFit(deg=1, use_xbias=False, init_w=[[[0.1]], [[-0.05]]], name='polyfit
↪')(input)
model = Model(inputs=input, outputs=output)

# predict without training (i.e. without gradient updates)
np.allclose(model.predict(random_xdata), random_ydata)
>>> True # True means prediction approx close enough
```

4.5.9 Mean and Variance Calculation Layer for Bayesian Neural Net

class astroNN.nn.layers.**FastMCInferenceMeanVar**(*name=None*, ***kwargs*)

Take mean and variance of the results of a TimeDistributed layer, assuming axis=1 is the timestamp axis

Returns A layer

Return type object

History

2018-Feb-02 - Written - Henry Leung (University of Toronto)
2018-Apr-13 - Update - Henry Leung (University of Toronto)

call(*inputs*, *training=None*)

Note Equivalent to __call__()

Parameters **inputs** (*tf.Tensor*) – Tensor to be applied

Returns Tensor after applying the layer

Return type tf.Tensor

get_config()

Returns Dictionary of configuration

Return type dict

If you wnat fast MC inference on GPU and you are using keras models, you should just use *FastMCInference*.

FastMCInferenceMeanVar is a layer designed to be used with Bayesian Neural Network with Dropout Variational Inference. *FastMCInferenceMeanVar* should be used with *FastMCInference* in general. The advantage of *FastMCInferenceMeanVar* layer is you can copy the data and calculate the mean and variance on GPU (if any) when you are doing dropout variational inference.

FastMCInferenceMeanVar can be imported by

```
from astroNN.nn.layers import FastMCInferenceMeanVar
```

It can be used with keras or tensorflow.keras, you just have to import the function from astroNN

```
def keras_model():
    # Your keras_model define here, assuming you are using functional API
    input = Input(.....)
    monte_carlo_dropout = FastMCInference(mc_num_here)
    # some layer here, you should use MCDropout from astroNN instead of Dropout from␣
↪Tensorflow:)
    result_mean_var = FastMCInferenceMeanVar()(previous_layer_here)
    return model

model.compile(loss=loss_func_here, optimizer=optimizer_here)

# Use the model to predict
output = model.predict(x)

# with dropout variational inference
# prediction and model uncertainty (variance) from the model
mean = output[0]
variance = output[1]
```

4.5.10 Repeat Vector Layer for Bayesian Neural Net

class astroNN.nn.layers.**FastMCRepeat**(*n*, *name=None*, ***kwargs*)

Prepare data to do inference, Repeats the input n times at axis=1

Parameters **n** (*int*) – Number of Monte Carlo integration

Returns A layer

Return type object

History

2018-Feb-02 - Written - Henry Leung (University of Toronto)
2018-Apr-13 - Update - Henry Leung (University of Toronto)

call(*inputs*, *training=None*)

Note Equivalent to __call__()

Parameters **inputs** (*tf.Tensor*) – Tensor to be applied

Returns Tensor after applying the layer which is the repeated Tensor

Return type tf.Tensor

get_config()

Returns Dictionary of configuration

Return type dict

If you wnat fast MC inference on GPU and you are using keras models, you should just use *FastMCInference*.

FastMCRepeat is a layer to repeat training data to do Monte Carlo integration required by Bayesian Neural Network.

FastMCRepeat is a layer designed to be used with Bayesian Neural Network with Dropout Variational Inference. *FastMCRepeat* should be used with *FastMCInferenceMeanVar* in general. The advantage of *FastMCRepeat* layer is you can copy the data and calculate the mean and variance on GPU (if any) when you are doing dropout variational inference.

FastMCRepeat can be imported by

```
from astroNN.nn.layers import FastMCRepeat
```

It can be used with keras or tensorflow.keras, you just have to import the function from astroNN

```
def keras_model():
    # Your keras_model define here, assuming you are using functional API
    input = Input(.....)
    monte_carlo_dropout = FastMCRepeat(mc_num_here)
    # some layer here, you should use MCDropout from astroNN instead of Dropout from␣
↪Tensorflow:)
    result_mean_var = FastMCInferenceMeanVar()(previous_layer_here)
    return model

model.compile(loss=loss_func_here, optimizer=optimizer_here)

# Use the model to predict
output = model.predict(x)

# with dropout variational inference
# prediction and model uncertainty (variance) from the model
```

(continues on next page)

(continued from previous page)

```
mean = output[0]
variance = output[1]
```

4.5.11 Fast Monte Carlo Integration Layer for Keras Model

class astroNN.nn.layers.**FastMCInference**(*n*, ***kwargs*)

Turn a model for fast Monte Carlo (Dropout, Flipout, etc) Inference on GPU

Parameters **n** (*int*) – Number of Monte Carlo integration

Returns A layer

Return type object

History 2018-Apr-13 - Written - Henry Leung (University of Toronto)

__call__(*model*)

Parameters **model** (*Union[keras.Model, keras.Sequential]*) – Keras model to be accelerated

Returns Accelerated Keras model

Return type Union[keras.Model, keras.Sequential]

get_config()

Returns Dictionary of configuration

Return type dict

FastMCInference is a layer designed for fast Monte Carlo Inference on GPU. One of the main challenge of MC integration on GPU is you want the data stay on GPU and you do MC integration on GPU entirely, moving data from drives to GPU is a very expensive operation. *FastMCInference* will create a new keras model such that it will replicate data on GPU, do Monte Carlo integration and calculate mean and variance on GPU, and get back the result.

Benchmark (Nvidia GTX1060 6GB): 98,000 7514 pixles APOGEE Spectra, traditionally the 25 forward pass spent ~270 seconds, by using *FastMCInference*, it only spent ~65 seconds to do the exact same task.

It can only be used with Keras model. If you are using customised model purely with Tensorflow, you should use *FastMCRepeat* and *FastMCInferenceMeanVar*

You can import the function from astroNN by

```
from astroNN.nn.layers import FastMCInference

# keras_model is your keras model with 1 output which is a concatenation of labels␣
↪prediction and predictive variance
keras_model = Model(....)

# fast_mc_model is the new keras model capable to do fast monte carlo integration on␣
↪GPU
fast_mc_model = FastMCInference(keras_model)

# You can just use keras API with the new model such as
result = fast_mc_model.predict(.....)

# here is the result dimension
predictions = result[:, :(result.shape[1] // 2), 0]  # mean prediction
```

(continues on next page)

(continued from previous page)

```
mc_dropout_uncertainty = result[:, :(result.shape[1] // 2), 1] * (self.labels_std **␣
↪2)  # model uncertainty
predictions_var = np.exp(result[:, (result.shape[1] // 2):, 0]) * (self.labels_std **␣
↪2)  # predictive uncertainty
```

4.5.12 Gradient Stopping Layer

class astroNN.nn.layers.**StopGrad**(*name=None*, *always_on=False*, ***kwargs*)

Stop gradient backpropagation via this layer during training, act as an identity layer during testing by default.

Parameters **always_on** (*bool*) – Default False which will on during train time and off during test time. True to enable it in every situation

Returns A layer

Return type object

History 2018-May-23 - Written - Henry Leung (University of Toronto)

call(*inputs*, *training=None*)

Note Equivalent to __call__()

Parameters **inputs** (*tf.Tensor*) – Tensor to be applied

Returns Tensor after applying the layer which is just the original tensor

Return type tf.Tensor

get_config()

Returns Dictionary of configuration

Return type dict

It uses `tf.stop_gradient` and acts as a Keras layer.

StopGrad can be imported by

```
from astroNN.nn.layers import StopGrad
```

It can be used with keras or tensorflow.keras, you just have to import the function from astroNN

```
def keras_model():
    # Your keras_model define here, assuming you are using functional API
    input = Input(.....)
    # some layers ...
    stopped_grad_layer = StopGrad()(...)
    # some layers ...
    return model
```

For example, if you have a model with multiple branches and you only want error backpropagate to one but not the other,

```
from astroNN.nn.layers import StopGrad
# we use zeros loss just to demonstrate StopGrad works and no error backprop from␣
↪StopGrad layer
from astroNN.nn.losses import zeros_loss
import numpy as np
from astroNN.shared.nn_tools import cpu_fallback
```

(continues on next page)

(continued from previous page)

```
from astroNN.config import keras_import_manager

keras = keras_import_manager()  # either import keras or tf.keras
cpu_fallback()  # force tf to use CPU

Input = keras.layers.Input
Dense = keras.layers.Dense
concatenate = keras.layers.concatenate
Model = keras.models.Model

# Data preparation
random_xdata = np.random.normal(0, 1, (100, 7514))
random_ydata = np.random.normal(0, 1, (100, 25))
input2 = Input(shape=[7514])
dense1 = Dense(100, name='normaldense')(input2)
dense2 = Dense(25, name='wanted_dense')(input2)
dense2_stopped = StopGrad(name='stopgrad', always_on=True)(dense2)
output2 = Dense(25, name='wanted_dense2')(concatenate([dense1, dense2_stopped]))
model2 = Model(inputs=input2, outputs=[output2, dense2])
model2.compile(optimizer=keras.optimizers.SGD(lr=0.1),
               loss={'wanted_dense2': 'mse', 'wanted_dense': zeros_loss})
weight_b4_train = model2.get_layer(name='wanted_dense').get_weights()[0]
weight_b4_train2 = model2.get_layer(name='normaldense').get_weights()[0]
model2.fit(random_xdata, [random_ydata, random_ydata])
weight_a4_train = model2.get_layer(name='wanted_dense').get_weights()[0]
weight_a4_train2 = model2.get_layer(name='normaldense').get_weights()[0]

print(np.all(weight_b4_train == weight_a4_train))
>>> True  # meaning all the elements from Dense with StopGrad layer are equal due to␣
↪no gradient update
print(np.all(weight_b4_train2 == weight_a4_train2))
>>> False  # meaning not all the elements from normal Dense layer are equal due to␣
↪gradient update
```

4.5.13 Boolean Masking Layer

class astroNN.nn.layers.**BoolMask**(*mask*, *name=None*, ***kwargs*)

Boolean Masking layer

Parameters **mask** (*np.ndarray*) – numpy boolean array as a mask for incoming tensor

Returns A layer

Return type object

History 2018-May-28 - Written - Henry Leung (University of Toronto)

call(*inputs*, *training=None*)

Note Equivalent to __call__()

Parameters **inputs** (*tf.Tensor*) – Tensor to be applied

Returns Tensor after applying the layer which is just the masked tensor

Return type tf.Tensor

get_config()

Returns Dictionary of configuration

Return type dict

BoolMask takes numpy boolean array as layer initialization and mask the input tensor.

BoolMask can be imported by

```
from astroNN.nn.layers import BoolMask
```

It can be used with keras or tensorflow.keras, you just have to import the function from astroNN

```
def keras_model():
    # Your keras_model define here, assuming you are using functional API
    input = Input(.....)
    # some layers ...
    stopped_grad_layer = BoolMask(mask=....)(...)
    # some layers ...
    return model
```

4.6 Callbacks and Utilities - astroNN.nn.callbacks, astroNN.nn.utilities

A callback is a set of functions to be applied at given stages of the training procedure. astroNN provides some customized callbacks which built on Keras and Tensorflow. Thus they are compatible with Keras with Tensorflow backend. You can just treat astroNN customized callbacks as conventional Keras callbacks.

astroNN also contains some handy utilities for data processing

4.6.1 Virtual CSVLogger (Callback)

class astroNN.nn.callbacks.**VirutalCSVLogger**(*filename='training_history.csv'*, *separator=','*, *append=False*)

A modification of keras' CSVLogger, but not actually write a file until you call method to save

Parameters

- **filename** (*str*) – filename of the log to be saved on disk
- **separator** (*str*) – separator of fields
- **append** (*bool*) – whether allow append or not

Returns callback instance

Return type object

History

2018-Feb-22 - Written - Henry Leung (University of Toronto)
2018-Mar-12 - Update - Henry Leung (University of Toronto)

savefile(*folder_name=None*)

the method to actually save the file to disk

Parameters **folder_name** (*Union[NoneType, str]*) – foldername, can be None to save to current directory

VirutalCSVLogger is basically Keras's CSVLogger without Python 2 support and won't write the file to disk until *savefile()* method is called after the training where Keras's CSVLogger will write to disk immediately.

VirutalCSVLogger can be imported by

```
from astroNN.nn.callbacks import VirutalCSVLogger
```

It can be used with Keras, you just have to import the function from astroNN

```
def keras_model():
    # Your keras_model define here
    return model

# Create a Virtual_CSVLogger instance first
csvlogger = VirutalCSVLogger()

# Default filename is training_history.csv
# You have to set filename first before passing to Keras
csvlogger.filename = 'training_history.csv'

model = keras_model()
model.compile(....)

model.fit(...,callbacks=[csvlogger])

# Save the file to current directory
csvlogger.savefile()

# OR to save the file to other directory
csvlogger.savefile(folder_name='some_folder')
```

4.6.2 Raising Error on Nan (Callback)

class astroNN.nn.callbacks.**ErrorOnNaN**

Callback that raise error when a NaN loss is encountered.

Returns callback instance

Return type object

History 2018-May-07 - Written - Henry Leung (University of Toronto)

ErrorOnNaN is basically Keras's TerminateOnNaN but will raise *ValueError* on Nan, its useful for python unittest to make sure you can catch the error and know something is wrong.

4.6.3 Normalizer (Utility)

astroNN *Normalizer* is called when *train()* method is called and involved *pre_training_checklist_master()* method defined in *NeuralNetMaster* Class. *Normalizer* will not normalize data/labels equal to `magicnumber` defined in configuration file. So that astroNN loss function can recognize those missing/bad data.

Normalizer consists of a few modes that you can, but the mode will minus mean and divide standard derivation to the data.

$$\text{Normalized Data} = \frac{\text{Data} - \text{Mean}}{\text{Standard Derivation}} \text{for Data} \neq \text{Magic Number}$$

1. *Mode 0* means normalizing data with mean=0 and standard derivation=1 (same as doing nothing)

```
# If we have some data
data = np.array([[1,2,3], [9,8,7]])

# THe normalized data, mean std will as follow by this mode
norm_data = array([[1,2,3], [9,8,7]])
# the mean and standard derivation used to do the normalization
mean = [0.]
std = [1.]
```

2. *Mode 1* means normalizing data with a single mean and a single standard derivation of the data

```
# If we have some data
data = np.array([[1,2,3], [9,8,7]])

# THe normalized data, mean std will as follow by this mode
norm_data = array([[-1.28653504, -0.96490128, -0.64326752], [ 1.28653504,  0.96490128,
↪  0.64326752]])
# the mean and standard derivation used to do the normalization
mean = [5.0]
std = [3.11]
```

3. *Mode 2* means normalizing data with pixelwise means and pixelwise standard derivations of the data

```
# If we have some data
data = np.array([[1,2,3], [9,8,7]])

# THe normalized data, mean std will as follow by this mode
norm_data = array([[-4., -3., -2.], [ 4.,  3.,  2.]])
# the mean and standard derivation used to do the normalization
mean = [5., 5., 5.]
std = [4., 3., 2.]
```

4. *Mode 3* means normalizing data with featurewise mean and standard derivation=1 the data (only centered the data), it is useful for normalizing spectra

```
# If we have some data
data = array([[1,2,3], [9,8,7]])

# THe normalized data, mean std will as follow by this mode
norm_data = array([[-1., -1., -1.], [ 1.,  1.,  1.]])
# the mean and standard derivation used to do the normalization
mean = [5., 5., 5.]
std = [1.]
```

5. *Mode 3s* means normalizing data with featurewise mean and standard derivation=1 the data (only centered the data), then apply sigmoid for normalization or sigmoid inverse for denormalization. It is useful for normalizing spectra for Variational Autoencoder with Negative Log Likelihood objective.

6. *Mode 255* means normalizing data with mean=127.5 and standard derivation=127.5, this mode is designed to normalize 8bit images

```
# If we have some data
data = np.array([[255,125,100], [99,87,250]])

# THe normalized data, mean std will as follow by this mode
norm_data = array([[ 1. , -0.01960784, -0.21568627], [-0.22352941, -0.31764706,  0.
↪96078431]])
```

(continues on next page)

(continued from previous page)

```
# the mean and standard derivation used to do the normalization
mean = [127.5]
std = [127.5]
```

You can set the mode from a astroNN neural net instance before called *train()* method by

```
# To set the normalization mode for input and labels
astronn_neuralnet.input_norm_mode = ...
astronn_neuralnet.labels_norm_mode = ...
```

You can use *Normalizer()* independently to take advantage of this function won't touch data equal `magicnumber`. *Normalizer()* always return you the normalized data, the mean and standard derivation used to do the normalization

```
from astroNN.nn.utilities.normalizer import Normalizer
import numpy as np

# Make some data up
data = np.array([[1.,2.,3.], [9.,8.,7.]])

# Setup a normalizer instance with a mode, lets say mode 1
normer = Normalizer(mode=1)

# Use the instance method normalize to normalize the data
norm_data = normer.normalize(data)

print(norm_data)
>>> array([[-1.28653504, -0.96490128, -0.64326752], [ 1.28653504,  0.96490128,  0.
↪64326752]])
print(normer.mean_labels)
>>> 5.0
print(normer.std_labels)
>>> 3.1091263510296048

# You can use the same instance (with same mean and std and mode) to demoralize data
denorm_data = normer.denormalize(data)

print(denorm_data)
>>> array([[1.,2.,3.], [9.,8.,7.]])
```

4.6.4 Useful Handy Tensorflow function - astroNN.nn

`astroNN.nn.`**`reduce_var`**(*x*, *axis=None*, *keepdims=False*)

Calculate variance using Tensorflow (as opposed to tf.nn.moment which return both variance and mean)

Parameters

- **x** (*tf.Tensor*) – Data
- **axis** (*int*) – Axis
- **keepdims** (*boolean*) – Keeping variance dimension as data or not

Returns Variance

Return type tf.Tensor

History 2018-Mar-04 - Written - Henry Leung (University of Toronto)

astroNN.nn.**intpow_avx2**(*x*, *n*)

Calculate integer power of float (including negative) even with Tensorflow compiled with AVX2 since –fast-math compiler flag aggressively optimize float operation which is common with AVX2 flag

Parameters

- **x** (*tf.Tensor*) – identifier
- **n** (*int*) – an integer power (a float will be casted to integer!!)

Returns powered float(s)

Return type tf.Tensor

History 2018-Aug-13 - Written - Henry Leung (University of Toronto)

```
from astroNN.nn import intpow_avx2
import tensorflow as tf

tf.enable_eager_execution()

print(intpow_avx2(tf.constant([-1.2]), 2))
>>> tf.Tensor([1.44], shape=(1,), dtype=float32)

print(tf.pow(tf.constant([-1.2]), 2))
# if your tensorflow is compiled with AVX2 or --fast-math
>>> tf.Tensor([nan], shape=(1,), dtype=float32)
# if your tensorflow is NOT compiled with AVX2 or --fast-math
>>> tf.Tensor([1.44], shape=(1,), dtype=float32)
```

4.6.5 NumPy Implementation of Tensorflow function - astroNN.nn.numpy

astroNN has some handy numpy implementation of a number of tensorflow functions. The list of available functions are

astroNN.nn.numpy.**kl_divergence**(*x*, *y*)

NumPy implementation of tf.distributions.kl_divergence

Either both x and y are ndarray or both x and y are astropy.Quatity, return without astropy units in all case

Parameters

- **x** (*Union[ndarray, float]*) – prediction
- **y** (*Union[ndarray, float]*) – ground truth

Returns KL-divergence

Return type Union[ndarray, float]

History 2018-May-13 - Written - Henry Leung (University of Toronto)

astroNN.nn.numpy.**l1**(*x*, *l1=0.0*)

NumPy implementation of tf.keras.regularizers.l1

Parameters

- **x** (*Union[ndarray, float]*) – Data to have L1 regularization coefficient calculated
- **l1** (*Union[ndarray, float]*) – L1 regularization parameter

Returns L1 regularization coefficient

Return type Union[ndarray, float]

History 2018-Apr-11 - Written - Henry Leung (University of Toronto)

astroNN.nn.numpy.**l2**(*x*, *l2=0.0*)

NumPy implementation of tf.keras.regularizers.l2

Parameters

- **x** (*Union[ndarray, float]*) – Data to have L2 regularization coefficient calculated
- **l2** (*Union[ndarray, float]*) – L2 regularization parameter

Returns L2 regularization coefficient

Return type Union[ndarray, float]

History 2018-Apr-11 - Written - Henry Leung (University of Toronto)

astroNN.nn.numpy.**mean_absolute_error**(*x*, *y*, *axis=None*)

NumPy implementation of tf.keras.metrics.mean_absolute_error with capability to deal with `magicnumber` and astropy Quantity

Either both x and y are ndarray or both x and y are astropy.Quatity, return without astropy units in all case

Parameters

- **x** (*Union[ndarray, float, astropy.Quatity]*) – prediction
- **y** (*Union[ndarray, float, astropy.Quatity]*) – ground truth
- **axis** (*Union[NoneType, int]*) – NumPy axis

Raise TypeError when only either x or y contains astropy units. Both x, y should carry/not carry astropy units at the same time

Returns Mean Absolute Error

Return type Union[ndarray, float]

History 2018-Apr-11 - Written - Henry Leung (University of Toronto)

astroNN.nn.numpy.**mean_absolute_percentage_error**(*x*, *y*, *axis=None*)

NumPy implementation of tf.keras.metrics.mean_absolute_percentage_error with capability to deal with `magicnumber` and astropy Quantity

Either both x and y are ndarray or both x and y are astropy.Quatity, return has no astropy units in all case

Parameters

- **x** (*Union[ndarray, float, astropy.Quatity]*) – prediction
- **y** (*Union[ndarray, float, astropy.Quatity]*) – ground truth
- **axis** (*Union[NoneType, int]*) – NumPy axis

Raise TypeError when only either x or y contains astropy units. Both x, y should carry/not carry astropy units at the same time

Returns Mean Absolute Percentage Error

Return type Union[ndarray, float]

History 2018-Apr-11 - Written - Henry Leung (University of Toronto)

astroNN.nn.numpy.**median_absolute_error**(*x*, *y*, *axis=None*)

NumPy implementation of a median version of tf.keras.metrics.mean_absolute_error with capability to deal with `magicnumber` and astropy Quantity

Either both x and y are ndarray or both x and y are astropy.Quatity, return without astropy units in all case

Parameters

- **x** (*Union[ndarray, float, astropy.Quatity]*) – prediction
- **y** (*Union[ndarray, float, astropy.Quatity]*) – ground truth
- **axis** (*Union[NoneType, int]*) – NumPy axis

Raise TypeError when only either x or y contains astropy units. Both x, y should carry/not carry astropy units at the same time

Returns Median Absolute Error

Return type Union[ndarray, float]

History 2018-May-13 - Written - Henry Leung (University of Toronto)

astroNN.nn.numpy.**median_absolute_percentage_error**(*x*, *y*, *axis=None*)

NumPy implementation of a median version of tf.keras.metrics.mean_absolute_percentage_error with capability to

deal with `magicnumber` and astropy Quantity

Either both x and y are ndarray or both x and y are astropy.Quatity, return has no astropy units in all case

Parameters

- **x** (*Union[ndarray, float, astropy.Quatity]*) – prediction
- **y** (*Union[ndarray, float, astropy.Quatity]*) – ground truth
- **axis** (*Union[NoneType, int]*) – NumPy axis

Raise TypeError when only either x or y contains astropy units. Both x, y should carry/not carry astropy units at the same time

Returns Median Absolute Percentage Error

Return type Union[ndarray, float]

History 2018-May-13 - Written - Henry Leung (University of Toronto)

astroNN.nn.numpy.**relu**(*x*)

NumPy implementation of tf.nn.relu

Parameters **x** (*Union[ndarray, float]*) – Data to have ReLU activated

Returns ReLU activated data

Return type Union[ndarray, float]

History 2018-Apr-11 - Written - Henry Leung (University of Toronto)

astroNN.nn.numpy.**sigmoid**(*x*)

NumPy implementation of tf.sigmoid, mask `magicnumber`

Parameters **x** (*Union[ndarray, float]*) – Data to be applied sigmoid activation

Returns Sigmoid activated data

Return type Union[ndarray, float]

History 2018-Apr-11 - Written - Henry Leung (University of Toronto)

astroNN.nn.numpy.**sigmoid_inv**(*x*)
NumPy implementation of tf.sigmoid inverse, mask magicnumber

Parameters **x** (*Union[numpy.ndarray, float]*) – Data to be applied inverse sigmoid activation

Returns Inverse Sigmoid activated data

Return type Union[numpy.ndarray, float]

History 2018-Apr-11 - Written - Henry Leung (University of Toronto)

4.7 Neural Nets Classes and Basic Usage - astroNN.models

4.7.1 Available astroNN Neural Net Classes

All astroNN Neural Nets are inherited from some child classes which inherited NeuralNetMaster, NeuralNetMaster also relies relies on two major component, *Normalizer* and *GeneratorMaster*

```
Normalizer (astroNN.nn.utilities.normalizer.Normalizer)

GeneratorMaster (astroNN.nn.utilities.generator.GeneratorMaster)
├── CNNDataGenerator
├── Bayesian_DataGenerator
└── CVAE_DataGenerator

NeuralNetMaster (astroNN.models.base_master_nn.NeuralNetMaster)
├── CNNBase
│   ├── ApogeeCNN
│   ├── StarNet2017
│   ├── SimplePloyNN
│   └── Cifar10CNN
├── BayesianCNNBase
│   ├── MNIST_BCNN  # For authors testing only
│   ├── ApogeeBCNNCensored
│   └── ApogeeBCNN
├── ConvVAEBase
│   └── APGOEECVAE  # For authors testing only
└── CGANBase
    └── GalaxyGAN2017  # For authors testing only
```

4.7.2 NeuralNetMaster Class API

All astroNN Neural Nets classes inherited from this astroNN.models.base_master_nn.NeuralNetMaster and thus methods of this class is shared across all astroNN Neural Nets classes.

class astroNN.models.base_master_nn.**NeuralNetMaster**
Top-level class for an astroNN neural network

Variables

- **name** – Full English name
- **_model_type** – Type of model
- **_model_identifier** – Unique model identifier, by default using class name as ID

- **_implementation_version** – Version of the model
- **_python_info** – Placeholder to store python version used for debugging purpose
- **_astronn_ver** – astroNN version detected
- **_keras_ver** – Keras version detected
- **_tf_ver** – Tensorflow version detected
- **currentdir** – Current directory of the terminal
- **folder_name** – Folder name to be saved
- **fullfilepath** – Full file path
- **batch_size** – Batch size for training, by default 64
- **autosave** – Boolean to flag whether autosave model or not
- **task** – Task
- **lr** – Learning rate
- **max_epochs** – Maximum epochs
- **val_size** – Validation set size in percentage
- **val_num** – Validation set autual number
- **beta_1** – Exponential decay rate for the 1st moment estimates for optimization algorithm
- **beta_2** – Exponential decay rate for the 2nd moment estimates for optimization algorithm
- **optimizer_epsilon** – A small constant for numerical stability for optimization algorithm
- **optimizer** – Placeholder for optimizer
- **targetname** – Full name for every output neurones

History

2017-Dec-23 - Written - Henry Leung (University of Toronto)
2018-Jan-05 - Updated - Henry Leung (University of Toronto)

flush()

Experimental, I don't think it works
Flush GPU memory from tensorflow

History 2018-Jun-19 - Written - Henry Leung (University of Toronto)

get_config()
Get model configuration as a dictionary

Returns dict

History 2018-May-23 - Written - Henry Leung (University of Toronto)

get_layer(**args*, ***kwargs*)
get_layer() method of tensorflow

get_weights()
Get all model weights

Returns weights arrays

Return type ndarray

History 2018-May-23 - Written - Henry Leung (University of Toronto)

has_model
Get whether the instance has a model, usually a model is created after you called train(), the instance will has no model if you did not call train()

Returns bool

History 2018-May-21 - Written - Henry Leung (University of Toronto)

hessian (*x=None*, *mean_output=False*, *mc_num=1*, *denormalize=False*, *method='exact'*)

Calculate the hessian of output to input

Please notice that the de-normalize (if True) assumes the output depends on the input data first orderly in which the hessians does not depends on input scaling and only depends on output scaling

The hessians can be all zeros and the common cause is you did not use any activation or activation that is still too linear in some sense like ReLU.

Parameters

- **x** (*ndarray*) – Input Data
- **mean_output** (*boolean*) – False to get all hessian, True to get the mean
- **mc_num** (*int*) – Number of monte carlo integration
- **denormalize** (*bool*) – De-normalize diagonal part of Hessian
- **method** (*str*) – Either 'exact' to calculate numerical Hessian or 'approx' to approximate Hessian from Jacobian

Returns An array of Hessian

Return type ndarray

History 2018-Jun-14 - Written - Henry Leung (University of Toronto)

hessian_diag (*x=None*, *mean_output=False*, *mc_num=1*, *denormalize=False*)

Calculate the diagonal part of hessian of output to input, avoids the calculation of the whole hessian and takes its diagonal

Please notice that the de-normalize (if True) assumes the output depends on the input data first orderly in which the diagonal part of the hessians does not depends on input scaling and only depends on output scaling

The diagonal part of the hessians can be all zeros and the common cause is you did not use any activation or activation that is still too linear in some sense like ReLU.

Parameters

- **x** (*ndarray*) – Input Data
- **mean_output** (*boolean*) – False to get all hessian, True to get the mean

- **mc_num** (*int*) – Number of monte carlo integration
- **denormalize** (*bool*) – De-normalize diagonal part of Hessian

Returns An array of Hessian

Return type ndarray

History 2018-Jun-13 - Written - Henry Leung (University of Toronto)

input_shape
Get input shape of the prediction model

Returns input shape expectation

Return type tuple

History 2018-May-21 - Written - Henry Leung (University of Toronto)

jacobian(*x=None*, *mean_output=False*, *mc_num=1*, *denormalize=False*)

Calculate jacobian of gradient of output to input high performance calculation update on 15 April 2018

Please notice that the de-normalize (if True) assumes the output depends on the input data first orderly in which the equation is simply jacobian divided the input scaling, usually a good approx. if you use ReLU all the way

Parameters

- **x** (*ndarray*) – Input Data
- **mean_output** (*boolean*) – False to get all jacobian, True to get the mean
- **mc_num** (*int*) – Number of monte carlo integration
- **denormalize** (*bool*) – De-normalize Jacobian

Returns An array of Jacobian

Return type ndarray

History

2017-Nov-20 - Written - Henry Leung (University of Toronto)
2018-Apr-15 - Updated - Henry Leung (University of Toronto)

jacobian_old(*x=None*, *mean_output=False*, *denormalize=False*)

Calculate jacobian of gradient of output to input

Please notice that the de-normalize (if True) assumes the output depends on the input data first orderly in which the equation is simply jacobian divided the input scaling

Parameters

- **x** (*ndarray*) – Input Data
- **mean_output** (*boolean*) – False to get all jacobian, True to get the mean
- **denormalize** (*bool*) – De-normalize Jacobian

History 2017-Nov-20 - Written - Henry Leung (University of Toronto)

output_shape
Get output shape of the prediction model

Returns output shape expectation

Return type tuple

History 2018-May-19 - Written - Henry Leung (University of Toronto)

plot_dense_stats()
Plot dense layers weight statistics

Returns A plot

History 2018-May-12 - Written - Henry Leung (University of Toronto)

plot_model(*name='model.png'*, *show_shapes=True*, *show_layer_names=True*, *rankdir='TB'*)
Plot model architecture with pydot and graphviz

Parameters

- **name** (*str*) – file name to be saved with extension, .png is recommended
- **show_shapes** (*bool*) – whether show shape in model plot
- **show_layer_names** (*bool*) – whether to display layer names
- **rankdir** (*bool*) – a string specifying the format of the plot, 'TB' for vertical or 'LR' for horizontal plot

Returns No return but will save the model architecture as png to disk

save(*name=None*, *model_plot=False*)
Save the model to disk

Parameters

- **name** (*string*) – Folder name to be saved
- **model_plot** (*boolean*) – True to plot model too

Returns A saved folder on disk

save_weights(*filename='model_weights.h5'*, *overwrite=True*)
Save model weights as .h5

Parameters

- **filename** (*str*) – Filename of .h5 to be saved
- **overwrite** (*bool*) – whether to overwrite

Returns None, a .h5 file will be saved

History 2018-May-23 - Written - Henry Leung (University of Toronto)

summary()
Get model summary

Returns None, just print

History 2018-May-23 - Written - Henry Leung (University of Toronto)

uses_learning_phase
To determine whether the model depends on keras learning flag. If False, then setting learning phase will not affect the model

Returns the boolean to indicate keras learning flag dependence of the model

Return type bool

History 2018-Jun-03 - Written - Henry Leung (University of Toronto)

CNNBase

Documented Members:

- *astroNN.models.apogee_models.ApogeeCNN()*
- *astroNN.models.apogee_models.StarNet2017()*
- astroNN.models.SimplePloyNN()

class astroNN.models.base_cnn.**CNNBase**
Top-level class for a convolutional neural network

evaluate(*input_data*, *labels*)
Evaluate neural network by provided input data and labels and get back a metrics score

Parameters

- **input_data** (*ndarray*) – Data to be inferred with neural network
- **labels** (*ndarray*) – labels

Returns metrics score dictionary

Return type dict

History 2018-May-20 - Written - Henry Leung (University of Toronto)

test(*input_data*)
Use the neural network to do inference

Parameters **input_data** (*ndarray*) – Data to be inferred with neural network

Returns prediction and prediction uncertainty

Return type ndarry

History 2017-Dec-06 - Written - Henry Leung (University of Toronto)

train(*input_data*, *labels*)
Train a Convolutional neural network

Parameters

- **input_data** (*ndarray*) – Data to be trained with neural network
- **labels** (*ndarray*) – Labels to be trained with neural network

Returns None

Return type NoneType

History 2017-Dec-06 - Written - Henry Leung (University of Toronto)

train_on_batch(*input_data*, *labels*)
Train a neural network by running a single gradient update on all of your data, suitable for fine-tuning

Parameters

- **input_data** (*ndarray*) – Data to be trained with neural network
- **labels** (*ndarray*) – Labels to be trained with neural network

Returns None

Return type NoneType

History 2018-Aug-22 - Written - Henry Leung (University of Toronto)

BayesianCNNBase

Documented Members:

- *astroNN.models.apogee_models.ApogeeBCNN()*
- *astroNN.models.apogee_models.ApogeeBCNNCensored()*

class astroNN.models.base_bayesian_cnn.**BayesianCNNBase**

Top-level class for a Bayesian convolutional neural network

History 2018-Jan-06 - Written - Henry Leung (University of Toronto)

evaluate(*input_data*, *labels*, *inputs_err=None*, *labels_err=None*)

Evaluate neural network by provided input data and labels and get back a metrics score

Parameters

- **input_data** (*ndarray*) – Data to be trained with neural network
- **labels** (*ndarray*) – Labels to be trained with neural network
- **inputs_err** (*Union([NoneType, ndarray])*) – Error for input_data (if any), same shape with input_data.
- **labels_err** (*Union([NoneType, ndarray])*) – Labels error (if any)

Returns metrics score dictionary

Return type dict

History 2018-May-20 - Written - Henry Leung (University of Toronto)

test(*input_data*, *inputs_err=None*)

Test model, High performance version designed for fast variational inference on GPU

Parameters

- **input_data** (*ndarray*) – Data to be inferred with neural network
- **inputs_err** (*Union([NoneType, ndarray])*) – Error for input_data, same shape with input_data.

Returns prediction and prediction uncertainty

History

2018-Jan-06 - Written - Henry Leung (University of Toronto)
2018-Apr-12 - Updated - Henry Leung (University of Toronto)

test_old(*input_data*, *inputs_err=None*)

Tests model, it is recommended to use the new test() instead of this deprecated method

Parameters

- **input_data** (*ndarray*) – Data to be inferred with neural network
- **inputs_err** (*Union([NoneType, ndarray])*) – Error for input_data, same shape with input_data.

Returns prediction and prediction uncertainty

History 2018-Jan-06 - Written - Henry Leung (University of Toronto)

train(*input_data*, *labels*, *inputs_err=None*, *labels_err=None*)
Train a Bayesian neural network

Parameters

- **input_data** (*ndarray*) – Data to be trained with neural network
- **labels** (*ndarray*) – Labels to be trained with neural network
- **inputs_err** (*Union([NoneType, ndarray])*) – Error for input_data (if any), same shape with input_data.
- **labels_err** (*Union([NoneType, ndarray])*) – Labels error (if any)

Returns None

Return type NoneType

History

2018-Jan-06 - Written - Henry Leung (University of Toronto)
2018-Apr-12 - Updated - Henry Leung (University of Toronto)

train_on_batch(*input_data*, *labels*, *inputs_err=None*, *labels_err=None*)
Train a Bayesian neural network by running a single gradient update on all of your data, suitable for fine-tuning

Parameters

- **input_data** (*ndarray*) – Data to be trained with neural network
- **labels** (*ndarray*) – Labels to be trained with neural network
- **inputs_err** (*Union([NoneType, ndarray])*) – Error for input_data (if any), same shape with input_data.
- **labels_err** (*Union([NoneType, ndarray])*) – Labels error (if any)

Returns None

Return type NoneType

History

2018-Aug-25 - Written - Henry Leung (University of Toronto)

ConvVAEBase

Documented Members:

- *astroNN.models.apogee_models.ApogeeCVAE()*

class astroNN.models.base_vae.**ConvVAEBase**
Top-level class for a Convolutional Variational Autoencoder

History 2018-Jan-06 - Written - Henry Leung (University of Toronto)

evaluate(*input_data*, *labels*)
Evaluate neural network by provided input data and labels/reconstruction target to get back a metrics score

Parameters

- **input_data** (*ndarray*) – Data to be inferred with neural network
- **labels** (*ndarray*) – labels

Returns metrics score

Return type float

History 2018-May-20 - Written - Henry Leung (University of Toronto)

test(*input_data*)
Use the neural network to do inference and get reconstructed data

Parameters **input_data** (*ndarray*) – Data to be inferred with neural network

Returns reconstructed data

Return type ndarry

History 2017-Dec-06 - Written - Henry Leung (University of Toronto)

test_encoder(*input_data*)
Use the neural network to do inference and get the hidden layer encoding/representation

Parameters **input_data** (*ndarray*) – Data to be inferred with neural network

Returns hidden layer encoding/representation

Return type ndarray

History 2017-Dec-06 - Written - Henry Leung (University of Toronto)

train(*input_data*, *input_recon_target*)
Train a Convolutional Autoencoder

Parameters

- **input_data** (*ndarray*) – Data to be trained with neural network
- **input_recon_target** (*ndarray*) – Data to be reconstructed

Returns None

Return type NoneType

History 2017-Dec-06 - Written - Henry Leung (University of Toronto)

train_on_batch(*input_data*, *input_recon_target*)
Train a AutoEncoder by running a single gradient update on all of your data, suitable for fine-tuning

Parameters

- **input_data** (*ndarray*) – Data to be trained with neural network
- **input_recon_target** (*ndarray*) – Data to be reconstructed

Returns None

Return type NoneType

History 2018-Aug-25 - Written - Henry Leung (University of Toronto)

4.7.3 Workflow of Setting up astroNN Neural Nets Instances and Training

astroNN contains some predefined neural networks which work well in certain aspect. For most general usage, I recommend you to create your own neural network for more flexibility and take advantage of astroNN custom loss function or layers.

For predefined neural network, generally you have to setup an instances of astroNN Neural Nets class with some predefined architecture. For example,

```
# import the neural net class from astroNN first
from astroNN.models import ApogeeCNN

# astronn_neuralnet is an astroNN's neural network instance
# In this case, it is an instance of ApogeeCNN
astronn_neuralnet = ApogeeCNN()
```

Lets say you have your training data prepared, you should specify what the neural network is outputing by setting up the *targetname*

```
# Just an example, if the training data is Teff, logg, Fe and absmag
astronn_neuralnet.targetname = ['teff', 'logg', 'Fe', 'absmag']
```

By default, astroNN will generate folder name automatically with naming scheme `astroNN_[month][day]_run[run number]`. But you can specify custom name by

```
# astronn_neuralnet is an astroNN's neural network instance
astronn_neuralnet.folder_name = 'some_custom_name'
```

You can enable autosave (save all stuffs immediately after training or save it yourself by

```
# To enable autosave
astronn_neuralnet.autosave = True

# To save all the stuffs, model_plot=True to plot models too, otherwise wont plot,␣
↪needs pydot_ng and graphviz
astronn_neuralnet.save(model_plot=False)
```

astroNN will normalize your data after you called *train()* method. The advantage of it is if you are using normalization provided by astroNN, you can make sure when *test()* method is called, the testing data will be normalized and prediction will be denormalized in the exact same way as training data. This can minimize human error.

If you want to normalize by yourself, you can disable it by

```
# astronn_neuralnet is an astroNN's neural network instance
astronn_neuralnet.input_norm_mode=0
astronn_neuralnet.labels_norm_mode = 0
```

You can add a list of Keras/astroNN callback by

```
astronn_neuralnet.callbacks = [# some callback(s) here)]
```

So now everything is set up for training

```
# Start the training
astronn_neuralnet.train(x_train,y_train)
```

If you did not enable autosave, you can save it after training by

```
# To save all the stuffs, model_plot=True to plot models too, otherwise wont plot,␣
↪needs pydot_ng and graphviz
astronn_neuralnet.save(model_plot=False)
```

4.7.4 Load astroNN Generated Folders

First way to load a astroNN generated folder, you can use the following code. You need to replace `astroNN_0101_run001` with the folder name. should be something like `astroNN_[month][day]_run[run number]`

`astroNN.models.load_folder`(*folder=None*)

To load astroNN model object from folder

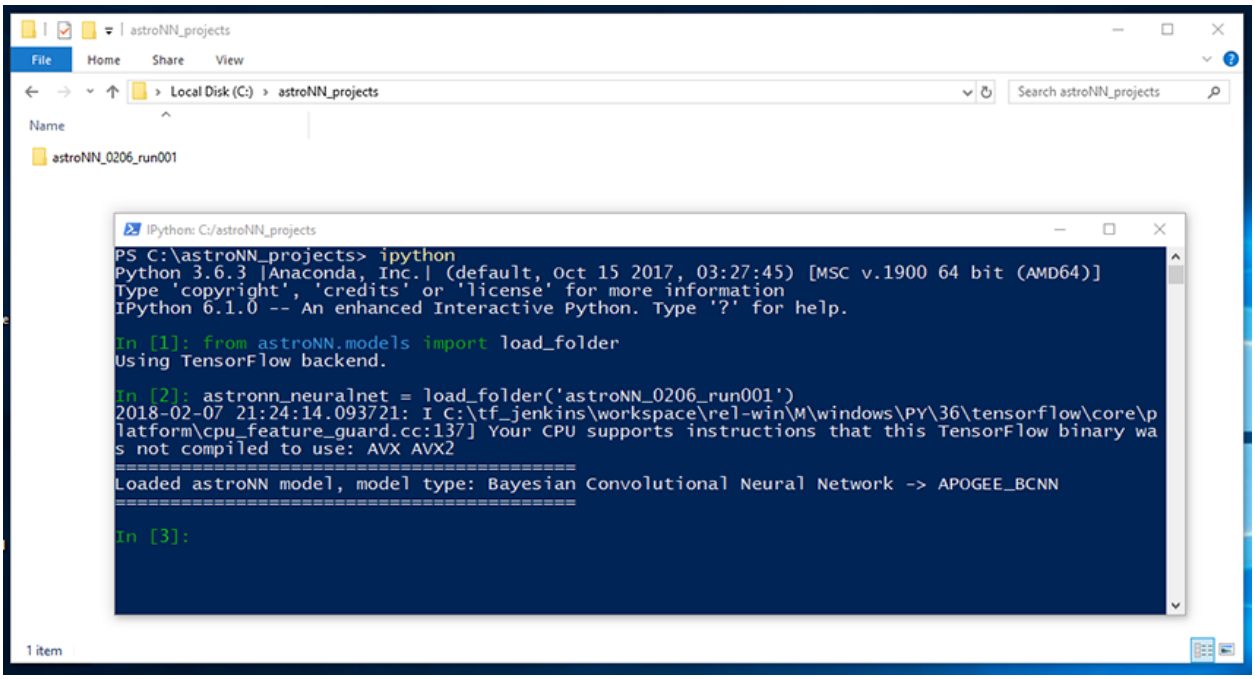
Parameters **folder** (*str*) – [optional] you should provide folder name if outside folder, do not specific when you are inside the folder

Returns astroNN Neural Network instance

Return type astroNN.nn.NeuralNetMaster.NeuralNetMaster

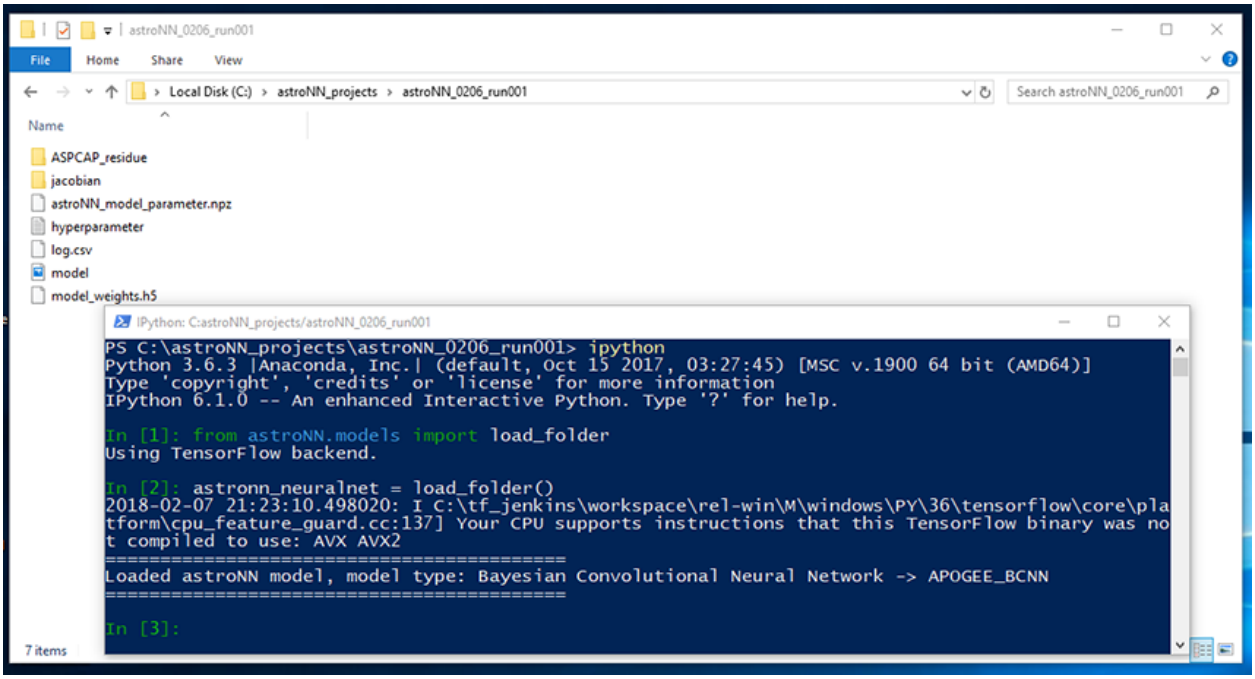
History 2017-Dec-29 - Written - Henry Leung (University of Toronto)

```
from astroNN.models import load_folder
astronn_neuralnet = load_folder('astroNN_0101_run001')
```



OR second way to open astroNN generated folders is to open the folder and run command line window inside there, or switch directory of your command line window inside the folder and run

```
from astroNN.models import load_folder
astronn_neuralnet = load_folder()
```



astronn_neuralnet will be an astroNN neural network object in this case. It depends on the neural network type which astroNN will detect it automatically, you can access to some methods like doing inference or continue the training (fine-tuning). You should refer to the tutorial for each type of neural network for more detail.

There is a few parameters from keras_model you can always access,

```
# The model summary from Keras
astronn_neuralnet.keras_model.summary()

# The model input
astronn_neuralnet.keras_model.input

# The model input shape expectation
astronn_neuralnet.keras_model.input_shape

# The model output
astronn_neuralnet.keras_model.output

# The model output shape expectation
astronn_neuralnet.keras_model.output_shape
```

astroNN neuralnet object also carries *targetname* (hopefully correctly set by the writer of neural net), parameters used to normalize the training data (The normalization of training and testing data must be the same)

```
# The tragetname corresponding to output neurone
astronn_neuralnet.targetname

# The model input
astronn_neuralnet.keras_model.input

# The mean used to normalized training data
astronn_neuralnet.input_mean_norm

# The standard derivation used to normalized training data
```

(continues on next page)

(continued from previous page)

```
astronn_neuralnet.input_std_norm

# The mean used to normalized training labels
astronn_neuralnet.labels_mean_norm

# The standard derivation used to normalized training labels
astronn_neuralnet.labels_std_norm
```

4.7.5 Load and Use Multiple astroNN Generated Folders

It is tricky to load and use multiple models at once since keras share a global session by default if no default tensorflow session provided and astroNN might encounter namespaces/scopes collision. So astroNN assign seperate Graph and Session for each astroNN neural network model. You can do:

```
from astroNN.models import load_folder

astronn_model_1 = load_folder("astronn_model_1")
astronn_model_2 = load_folder("astronn_model_2")
astronn_model_3 = load_folder("astronn_model_3")

with astronn_model_1.graph.as_default():
    with astronn_model_1.session.as_default():
        # do stuff with astronn_model_1 here

with astronn_model_2.graph.as_default():
    with astronn_model_2.session.as_default():
        # do stuff with astronn_model_2 here

with astronn_model_3.graph.as_default():
    with astronn_model_3.session.as_default():
        # do stuff with astronn_model_3 here

# For example do things with astronn_model_1 again
with astronn_model_1.graph.as_default():
    with astronn_model_1.session.as_default():
        # do more stuff with astronn_model_1 here
```

4.7.6 Workflow of Testing and Distributing astroNN Models

The first step of the workflow should be loading an astroNN folder as described above.

Lets say you have loaded the folder and have some testing data, you just need to provide the testing data without any normalization if you used astroNN normalization during training. The testing data will be normalized and prediction will be denormalized in the exact same way as training data.

```
# Run forward pass for the test data throught the neural net to get prediction
# The prediction should be denormalized if you use astroNN normalization during␣
↪training
prediction = astronn_neuralnet.test(x_test)
```

You can always train on new data based on existing weights

```
# Start the training on existing models (fine-tuning), astronn_neuralnet is a trained
↪astroNN models
astronn_neuralnet.train(x_train,y_train)
```

4.7.7 Creating Your Own Model with astroNN Neural Net Classes

You can create your own neural network model inherits from astroNN Neural Network class to take advantage of the existing code in this package. Here we will go thought how to create a simple model to do classification with MNIST dataset with one convolutional layer and one fully connected layer neural network.

Lets create a python script named `custom_models.py` under an arbitrary folder, lets say `~/` which is your home folder, add `~/custom_models.py` to astroNN configuration file.

```
# import everything we need

# astroNN keras_import_manager will import tf.keras automatically if keras is not
↪detected
from astroNN.config import keras_import_manager
# this is the astroNN neural net abstract class we will going to inherit from
from astroNN.models.CNNBase import CNNBase

keras = keras_import_manager()
regularizers = keras.regularizers
MaxPooling2D, Conv2D, Dense, Flatten, Activation, Input = keras.layers.MaxPooling2D,
↪keras.layers.Conv2D, \
                                                         keras.layers.Dense, keras.
↪layers.Flatten, \
                                                         keras.layers.Activation,
↪keras.layers.Input

# now we are creating a custom model based on astroNN neural net abstract class
class my_custom_model(CNNBase):
    def __init__(self, lr=0.005):
        # standard super for inheriting abstract class
        super().__init__()

        # some default hyperparameters
        self._implementation_version = '1.0'
        self.initializer = 'he_normal'
        self.activation = 'relu'
        self.num_filters = [8]
        self.filter_len = (3, 3)
        self.pool_length = (4, 4)
        self.num_hidden = [128]
        self.max_epochs = 1
        self.lr = lr
        self.reduce_lr_epsilon = 0.00005

        self.task = 'classification'
        # you should set the targetname some that you know what those output neurones
↪are representing
        # in this case the outpu the neurones are simply representing digits
        self.targetname = ['Zero', 'One', 'Two', 'Three', 'Four', 'Five', 'Six',
↪'Seven', 'Eight', 'Nine']

        # set default input norm mode to 255 to normalize images correctly
```

(continues on next page)

(continued from previous page)

```
        self.input_norm_mode = 255
        # set default labels norm mode to 0 (equivalent to do nothing) to normalize␣
↪labels correctly
        self.labels_norm_mode = 0

    def model(self):
        input_tensor = Input(shape=self._input_shape, name='input')
        cnn_layer_1 = Conv2D(kernel_initializer=self.initializer, padding="same",␣
↪filters=self.num_filters[0],
                             kernel_size=self.filter_len)(input_tensor)
        activation_1 = Activation(activation=self.activation)(cnn_layer_1)
        maxpool_1 = MaxPooling2D(pool_size=self.pool_length)(activation_1)
        flattener = Flatten()(maxpool_1)
        layer_2 = Dense(units=self.num_hidden[0], kernel_initializer=self.
↪initializer)(flattener)
        activation_2 = Activation(activation=self.activation)(layer_2)
        layer_3 = Dense(units=self.labels_shape, kernel_initializer=self.
↪initializer)(activation_2)
        output = Activation(activation=self._last_layer_activation, name='output
↪')(layer_3)

        model = Model(inputs=input_tensor, outputs=output)

        return model
```

Save the file and we can open python under the same location as the python script

```
# import everything we need
from custom_models import my_custom_model
from keras.datasets import mnist
from keras import utils

# load MNIST
(x_train, y_train), (x_test, y_test) = mnist.load_data()
# convert to approach type
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
y_train = utils.to_categorical(y_train, 10)

# create a neural network instance
net = my_custom_model()

# train
net.train(x_train, y_train)

# save the model after training
net.save("trained_models_folder")
```

If you want to share the trained models, you have to copy `custom_models.py` to the inside of the folder so that astroNN can load it successfully on other computers.

The second way is you send the file which is `custom_models.py` to the target computer and install the file by adding the file to `config.ini` on the target computer.

You can simply load the folder on other computers by running python inside the folder and run

```
# import everything we need
```

(continues on next page)

(continued from previous page)

```
from astroNN.models import load_folder

net = load_folder()
```

OR outside the folder `trained_models_folder`

```
# import everything we need
from astroNN.models import load_folder

net = load_folder("trained_models_folder")
```

4.7.8 NeuralNetMaster Class

NeuralNetMaster is the top level abstract class for all astroNN sub neural network classes. NeuralNetMaster define the structure of how an astroNN neural network class should look like.

NeuralNetMaster consists of a pre-training checking (check input and labels shape, cpu/gpu check and create astroNN folder for every run.

CHAPTER 5

Neural Net Introduction and Demonstration

5.1 Bayesian Neural Net with Dropout Variational Inference

With traditional neural network, weight in neural network are point estimate which result a point estimate result. Unlike statistical modelling which have uncertainty estimates, the whole point of machine learning is just learn from data and predict an single outcome. Uncertainty estimates is important in astronomy and it will be best if we could add uncertainty to neural network.

5.1.1 Background Knowledge

To understand Bayesian Neural Net, we first need to understand some background knowledge.

Bayes Rule

To understand how a Bayesian Neural Net works, we must first known about Bayesian statistics. The core of Bayesian statistic is Bayes Rule.

Suppose we have event A and B. Bayes Rule tells us $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$ where $P(A|B)$ is conditional probability which represents the likelihood of event A occurring given that B occurred. $P(B|A)$ represents the likelihood of event B occurring given that A occurred. $P(A)$ and :math‘P(B)‘ are probability of observing A and B independently of each other.

The Bayesian interpretation of a probablility is a measure of a prior belief. In such case, $P(A)$ can be viewed as a prior belief in A and $P(A|B)$ measures the postterior belief of having accounted for B.

Simple Bayesian Regression

The problem is a linear regression problem, we have some input data X and output data Y and we want to find w such that $Y = wX$. Suppose we use Mean Squared Error (L2) loss which is commonly found in neural network. The objective $(Y - wX)^2$

First step, we need to somehow change this to a probability. You want to maximizing the likelihood to generate Y given you have X and w, i.e. $P(Y|X,w)$

Please notice using Mean Squared Error (L2), it is equivalent maximizing the log-likelihood of a Gaussian, i.e Y is Gaussian distributed.

But we want this problem be Bayesian, so we impose a prior belief on our weight, $P(Y|X,w)P(w)$. Usually we set gaussian distribution as our belief.

By Bayes Rule, the posterior distribution of the weight is $P(w|X,Y) = \frac{P(Y|X,w)P(w)}{C}$ and C is $P(Y)$ or $\int P(X,w)dw$, an integral usually very difficult to calculate.

Variational Inference

To solve this problem we will need to use Variational Inference. How to do Variational Inference.

The first step we need to introduce a parameterised distribution $Q(w|v)$, Q representing a variational distribution and v is the variational parameter, over w to approximate the true posterior.

And bingo, another advantage is from an integration problem, we now have an optimizing problem on variational parameter v. What are we optimizing to? We need to have a v so that to match the true posterior distribution as good as possible. True posterior refers to $P(w|y,x)$ and of course we better have a $Q(w|v)$ which close to the true posterior.

Approximation to the integral of a probability distribution ($\int P(X,w)dw$ in this case) can be done by Monte Carlo Sampling (similarilty to estimation of π by MC sampling)

5.1.2 Dropout Variational Inference

The core idea Bayesian Neural Network is Neural Net with Dropout Variational Inference and gaussian prior weights is bayesian. By reparametrising the approximate variational distribution *Q(w|v)* to be Bernoulli

$$r_i = \text{Bernoulli}(p)$$
$$\hat{y}_i = r_i * y_i$$

which is exactly the thing used by dropout.

Thus the loss is

$$\mathcal{L}_{dropout} = \frac{1}{D}\sum_{i=1}^{batch}(Loss_i) + \lambda\sum_{i=1}^{Layer}(Weight)^2$$

5.1.3 How is uncertainty calculated from neural network for regression task

$$\text{Prediction} = \text{Mean from Dropout Variational Inference}$$

Total Variance = Variance from Dropout Variational Inference + Mean of Predictive Variance Output + Inverse Model Precision

Or if you have known input data uncertainty, you should add the propagated uncertainty to the final variance too.

The final prediction will be

$$\text{Prediction with Error} = \text{Prediction} \pm \sqrt{\text{Total Variance}}$$

Inverse Model Precision is by definition

$\tau^{-1} = \frac{2N\lambda}{l^2 p}$where λ is the l2 regularization parameter, l is scale length, p is the probability of a neurone NOT being dropped and N is t

For more detail, please see my demonstration here

5.1.4 A simple way to think about predictive, model and propagated uncertainty

Since Bayesian Neural Network involves multiple source of uncertainty and they can be confusing. There is one simple way to think about these uncertainty.

Let's say you have a student and some maths problems with solutions and some maths problems without solutions. For simplicity all the maths problems are only either differentiation or integration. You want the solution for those maths problems without solution. One way to do it is to let the student to do the maths with known solution, and evaluate his/her performance. If the student did all the integration problems wrong, then you know the integration solutions from the student cannot be trusted.

In more real life situation, you don't know the training process/data, but you can interact with a trained student. Now you just give an integration problem to the student, the student should tells you he/she does not have confidence on that problem at all because it is about integration and the student knows his/her own ability for doing integration poorly. This is something that is predictable, so we call them predictive uncertainty.

Let's say the student has done very well on differentiation problems and you should expect he/she has a high confidence on this area. But if you are a teacher, you know if students said they understand a topic, they probably not really understand it. One way to measure the model uncertainty from the student is you give the problems to the student to solve and you get back a set of solutions. And after a week or so, you give the same problems to the student to solve and you get another set of solutions. If the two solutions are the same, and the student said he/she is confident, then you know the solutions are probably right. If the two solutions are not the same, then even the student said he/she is confident, you should not trust those solutions from the student.

The propagated uncertainty can be just as simple as you have some typos in the problems, and lead to the student giving some wrong answers.

5.2 Gaia DR2 with astroNN result

Gaia DR2 is released on 25 April 2018 with data collected from 25 July 2014 to 23 May 2016 with 1.5 billion sources.

Official Gaia DR2 page: https://www.cosmos.esa.int/web/gaia/dr2

astroNN is used to train neural network with Gaia DR1 parallax to predict intrinsic brightness of stars from APOGEE spectra. Since Gaia uses geometric method to infer distances to stars, and it has its own limitation, the major one will be the star must be close to us. If neural network can infer intrinsic brightness based on APOGEE spectra, with apparent magnitude we can get the distance as long as we have the stellar spectra.

This page will act as a notebook for the author (Henry) and share his latest update on Gaia DR2 preparation.

FAQ: What is fakemag? : http://astronn.readthedocs.io/en/latest/tools_gaia.html#fakemag-dummy-scale

FAQ: Which band I will use for apparent magnitude?: K-mag will be used to minimize the effect of extinction

5.2.1 (25 Apr 2018 update) Neural Network Distance Prediction on the whole APOGEE DR14 result with Gaia DR2

Procedure to reproduce the result is described here: https://github.com/henrysky/astroNN/tree/master/demo_tutorial/gaia_dr1_dr2/

Neural Network trained only Gaia DR1 (20% parallax error cuts)-APOGEE DR14 (SNR>50, STARFLAG==0) overlap, around 12,000 spectra. Results are expressed in mean absolute percentage error. Gaia DR2 refers to the subset of DR2 matched with Apogee DR14, parallax > 0 and parallax error < 0.2

Outperformed Apogee Distances DR14 BPG Catalog:

- Apogee Distances DR14 BPG (20% Model Confidence Cut): 77,401 spectra - 20.6%

- `astroNN ApogeeBCNN` (20% Neural Network Confidence Cut): 57,704 spectra - 14.5%
- `astroNN ApogeeBCNN` (25% Neural Network Confidence Cut): 76,136 spectra - 16.8%
- `astroNN ApogeeBCNN` (100% Neural Network Confidence Cut): 92,887 spectra - 22.6%

Outperformed "teacher" Gaia DR1 with 20% error cuts slightly on training set spectra:

- `astroNN ApogeeBCNN` (20% Neural Network Confidence Cut): 10,039 spectra - 6.74% mean absolute percentage error with DR2
- Gaia DR1 (20% error cuts): 9,019 spectra - 6.97% mean absolute percentage error with DR2

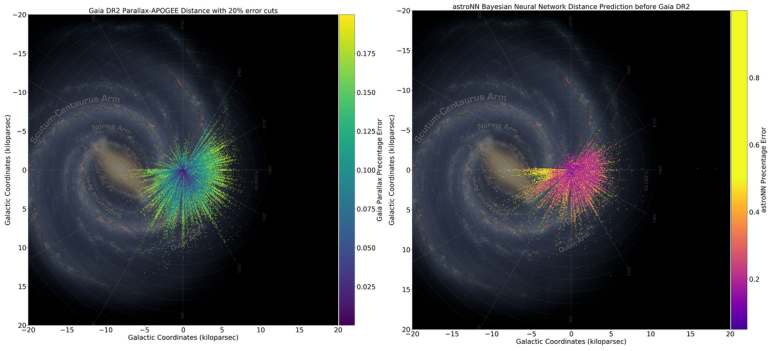
Gaia DR1, Anderson2017 with 20% error cuts in APOGEE DR14 crossed matched:

- Gaia DR1 (20% Observation Error Cut): 20,675 spectra - 8.3% mean absolute percentage error with DR2
- Anderson2017 (20% Model Confidence Cut): 25,303 spectra - 8.4% mean absolute percentage error with DR2

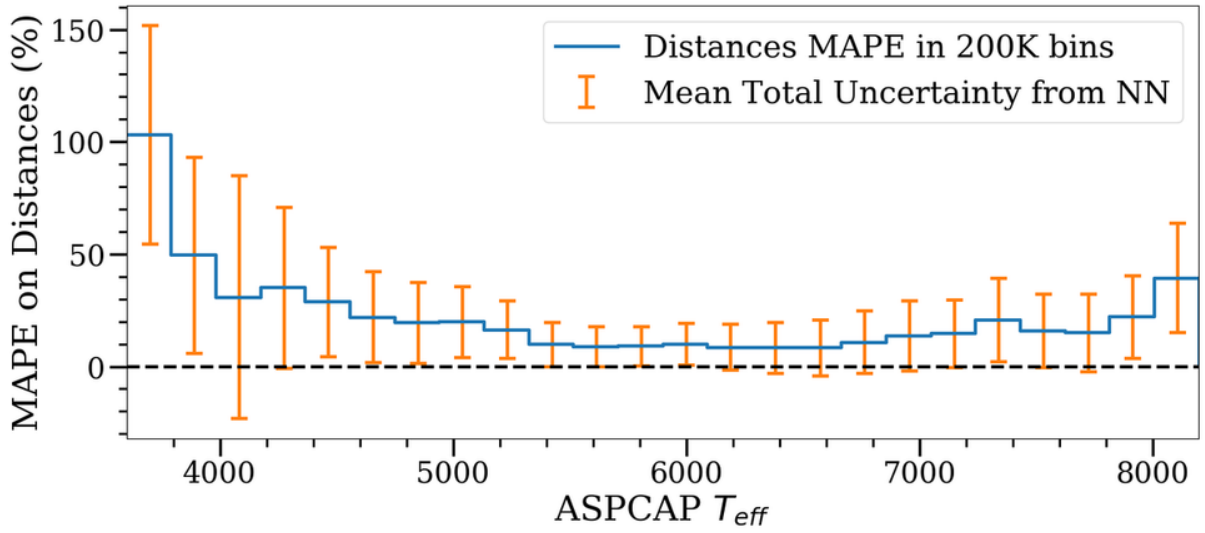
Apogee Red Clumps - astroNN - Gaia DR2 crossed matched, Red Clumps Catalog DR14 is better than NN:

- The whole Red Clumps Catalog: 22,421 spectra - 20.6% mean absolute percentage error with DR2
- Red Clumps Catalog crossed matched: 12,476 spectra - 18.9% mean absolute percentage error with DR2
- astroNN crossed matched: 12,476 spectra - 25.0% mean absolute percentage error with DR2

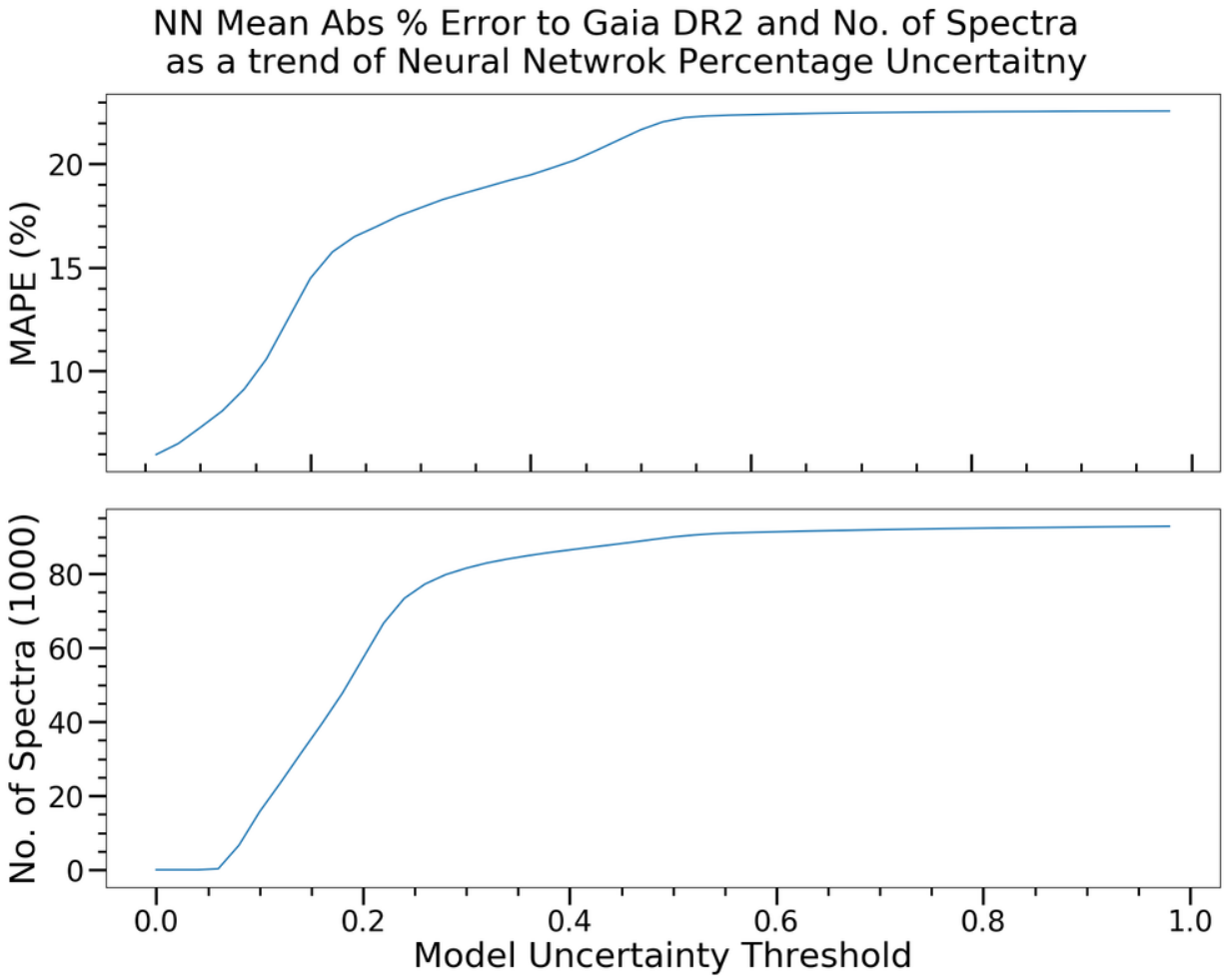
Internal model identifier for the author: `astroNN_0422_run001`



Neural Network Mean Absolute Percentage Error to Gaia DR2 as a function of Teff



Neural Network Mean Absolute Percentage Error to Gaia DR2 as a function of neural network uncertainty estimation



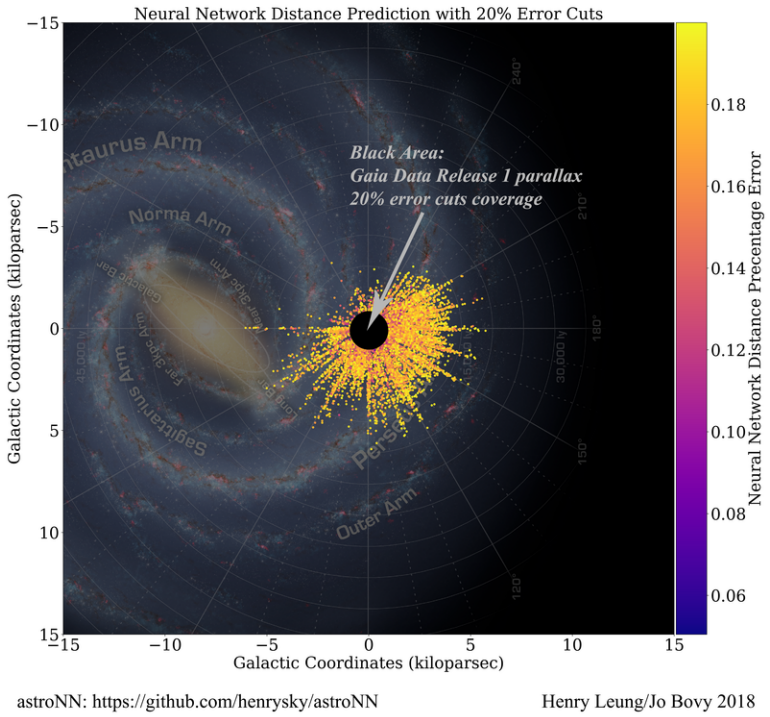
5.2.2 Plans/Questions

1. Train neural network on Gaia DR1 and validate on Gaia DR2 (result stated above)
2. Temperature cuts on spectra? (Didn't do it)
3. If neural network turns out very accurate when DR2 comes out, how did neural network predict those distance?
4. If neural network turns out very accurate when DR2 comes out, then we can get distance for many APOGEE spectra?
5. (No Need, the result is pretty good) If neural network failed, is predicting intrinsic brightness from APOGEE spectra impossible, or just because the training set is too small in DR1 led to failure?

5.2.3 Nerual Network Distance Prediction on the whole APOGEE DR14

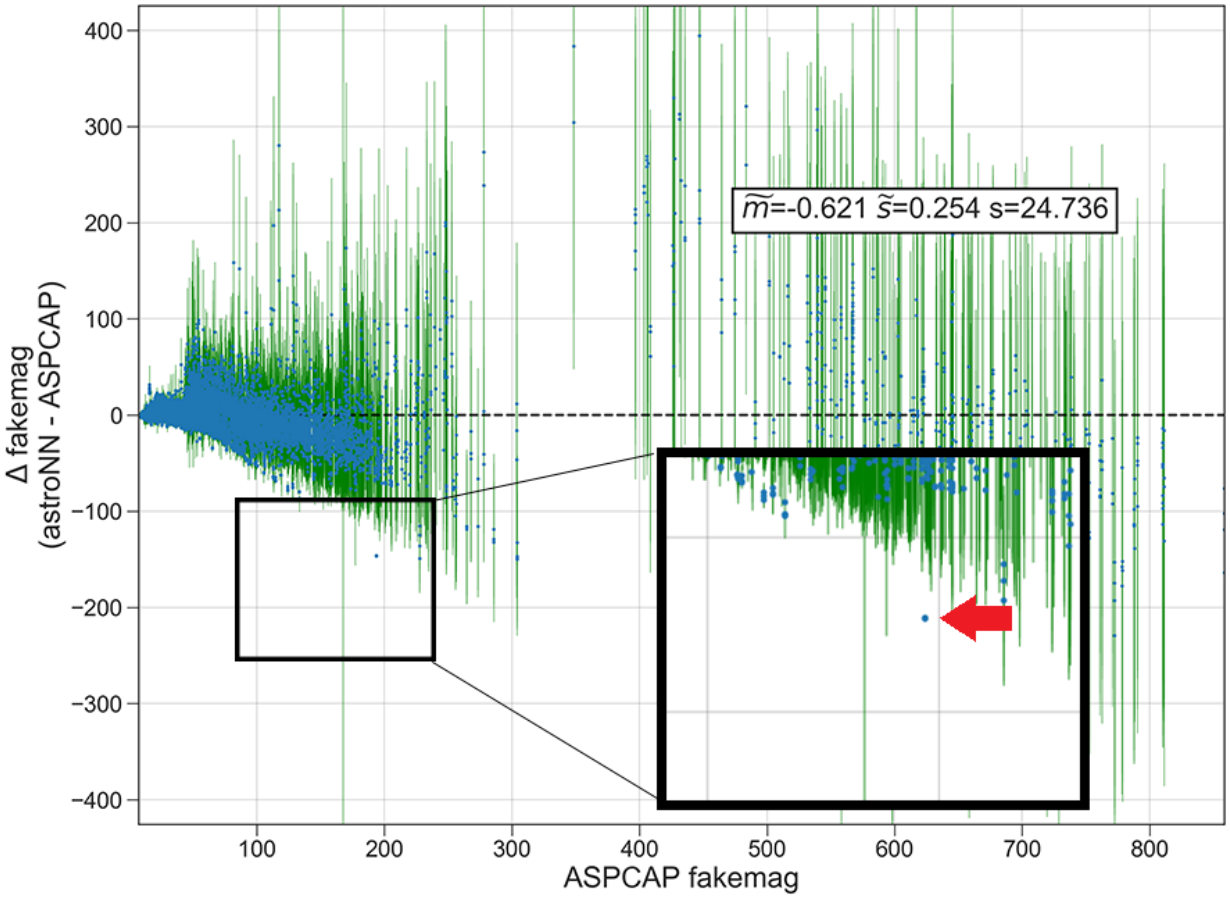
Neural Network trained only Gaia DR1 (20% parallax error cuts)-APOGEE DR14 (SNR>50, STARFLAG==0) overlap

Testing on the whole APOGEE DR14 (SNR>50, STARFLAG==0 cuts), around ~120,000 spectra



5.2.4 2M16363993+3654060 Distance Disagreement between astroNN and Gaia/Anderson2017 Parallax

Internal model identifier for the author: `astroNN_0128_run002`



Neural Network trained on Anderson2017 parallax constantly predicted an almost constant offset with very small uncertainty to the ground truth (Anderson2017) on the star with APOGEE_ID *2M16363993+3654060*. astroNN agreed pretty well with APOGEE_distances BPG_dist50. Seems like Gaia/Anderson2017 are the one which is far off.

I have to emphasise that the neural network is trained on the parallax from Anderson2017 which is improved parallax from Gaia DR1. There is no surprise that neural network identified outliers from the training/testing set. But the fact that neural network managed to have a similar answer with *APOGEE_distances BPG_dist50* may indicate neural network learned some "correct" physics to infer intrinsic distance from APOGEE spectra.

The result:

1. astroNN Bayesian Neural Network[1] : 2287.61 parsec $\pm$ 107.27 parsec
2. APOGEE_distances BPG_dist50[2] : 2266.15 parsec $\pm$ 266.1705 parsec
3. Anderson2017 parallax: 568.08 parsec $\pm$ 403.86 parsec
4. Gaia DR1 parallax: 318.05 parsec $\pm$ 1021.73 parsec

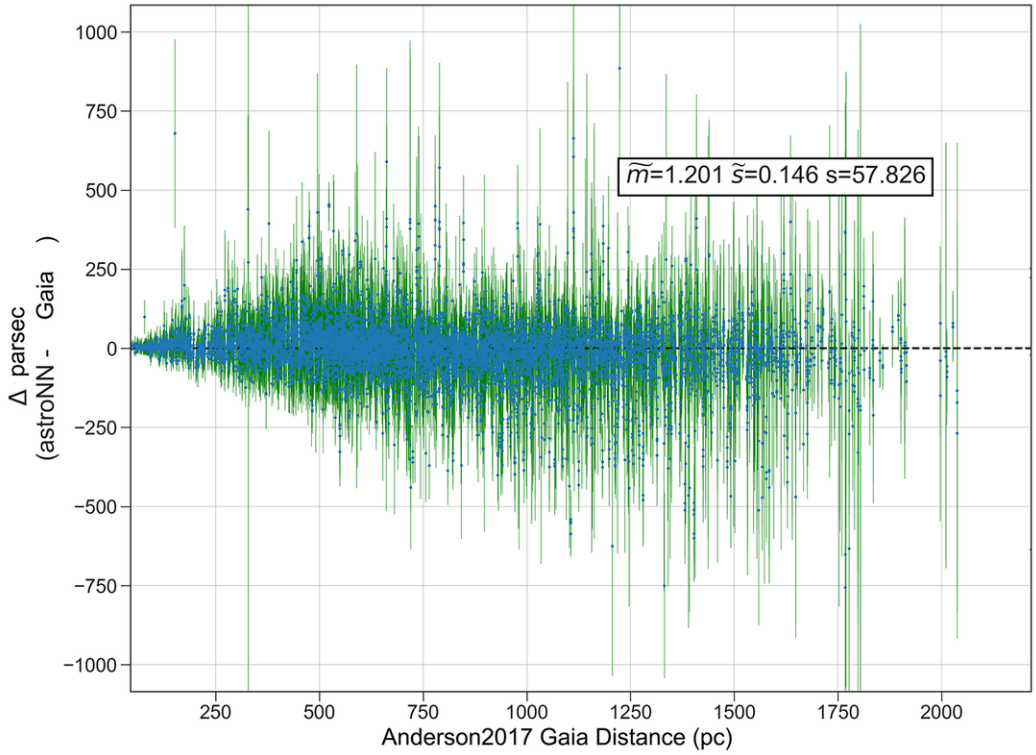
5.2.5 Distance Prediction with APOGEE-North Spectra

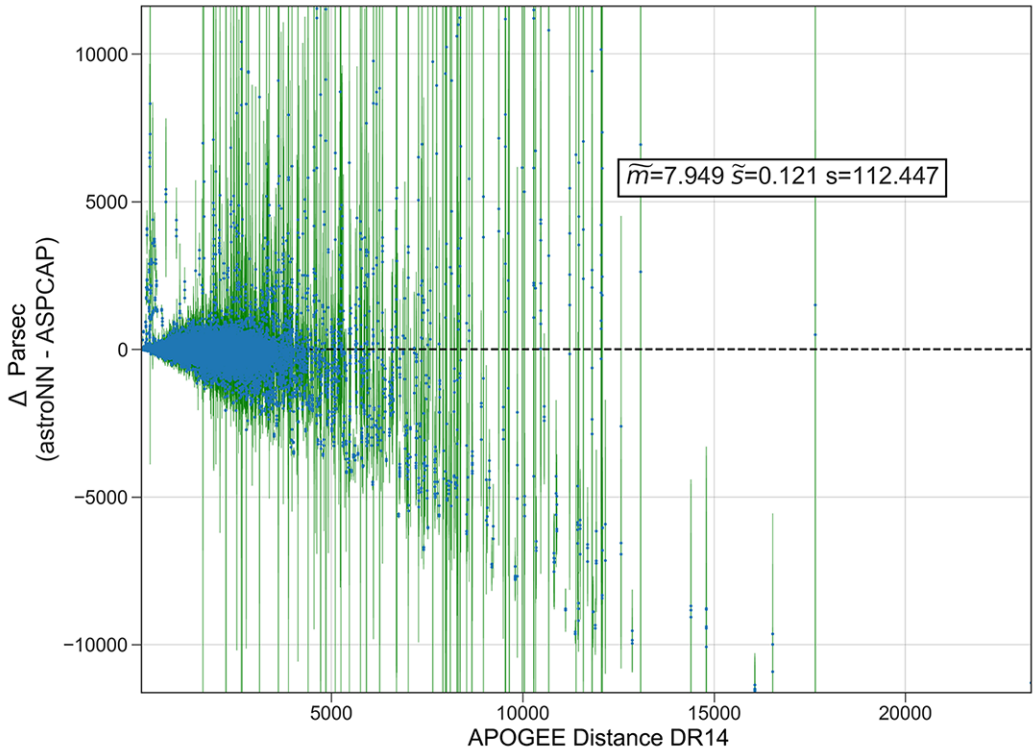
Internal model identifier for the author: `astroNN_0224_run002`

By using *astroNN.models.Apogee_BCNN* to train a neural network on Anderson2017 improved Gaia parallax (Predict stellar intrinsic brightness from their spectra). Here is the result

[1] Trained on ASPCAP parameters [Teff, Log(g) and 22 abundances] and Anderson2017 parallax
[2] http://www.sdss.org/dr14/data_access/value-added-catalogs/?vac_id=apogee-dr14-based-distance-estimations

1. First image, Anderson2017 is the ground truth and tested the neural network on individual spectra
2. Second image, assume APOGEE Distances DR14 is the ground truth, tested the neural network on individual spectra

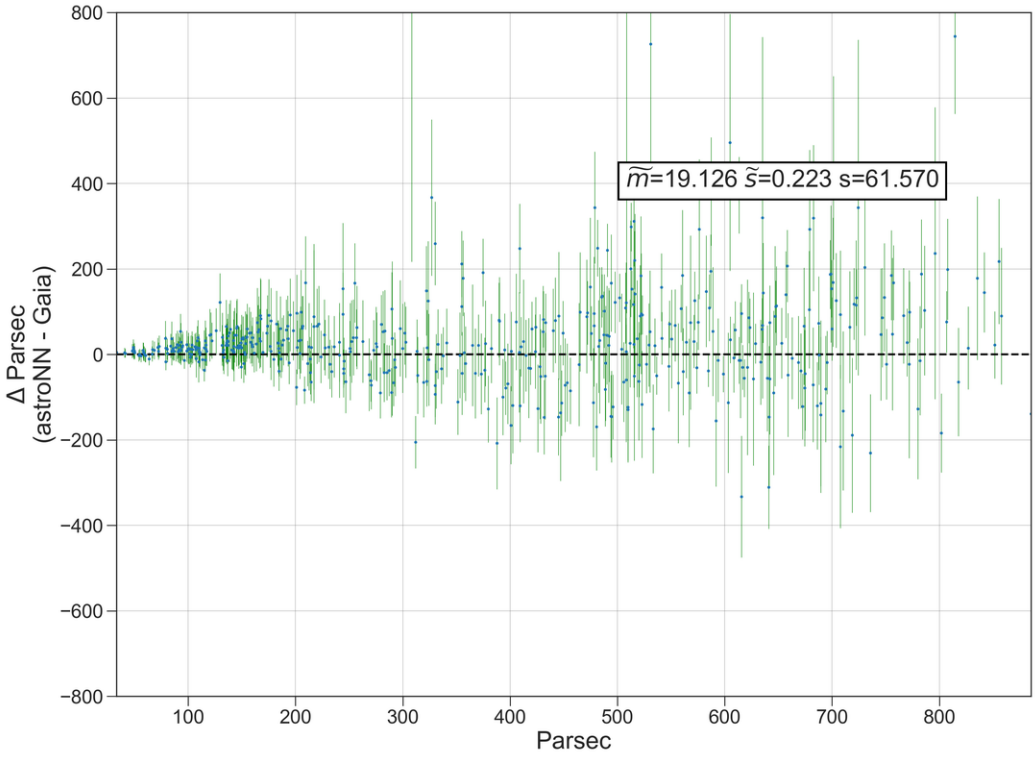




5.2.6 Distance Prediction with APOGEE-South Spectra

Internal model identifier for the author: `astroNN_0224_run002`

The neural network has trained on APOGEE-North spectra and gaia parallax. And then neural network has been tested on spectra from APOGEE-South (Different telescope and cameras)

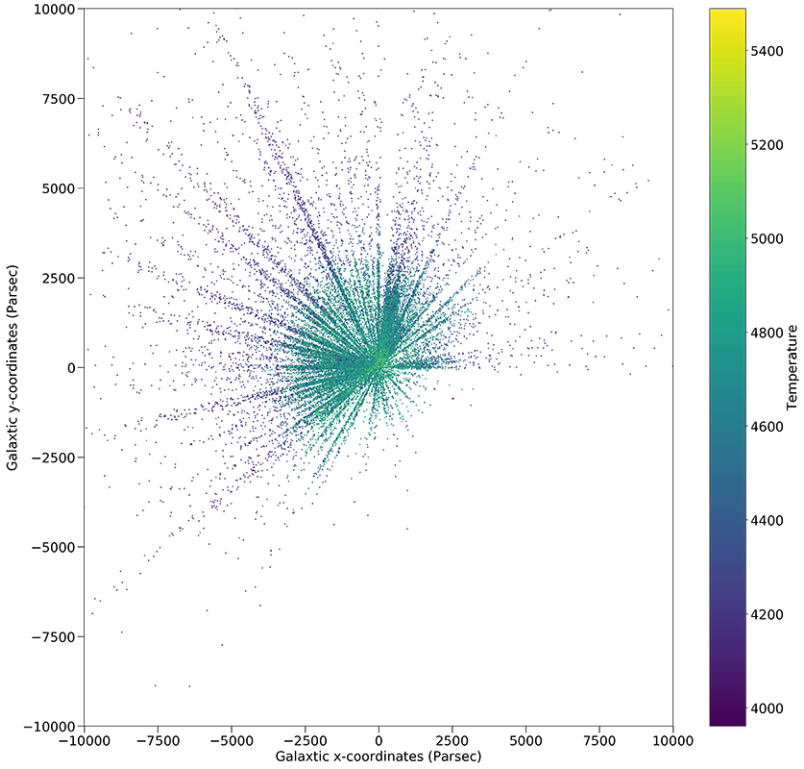


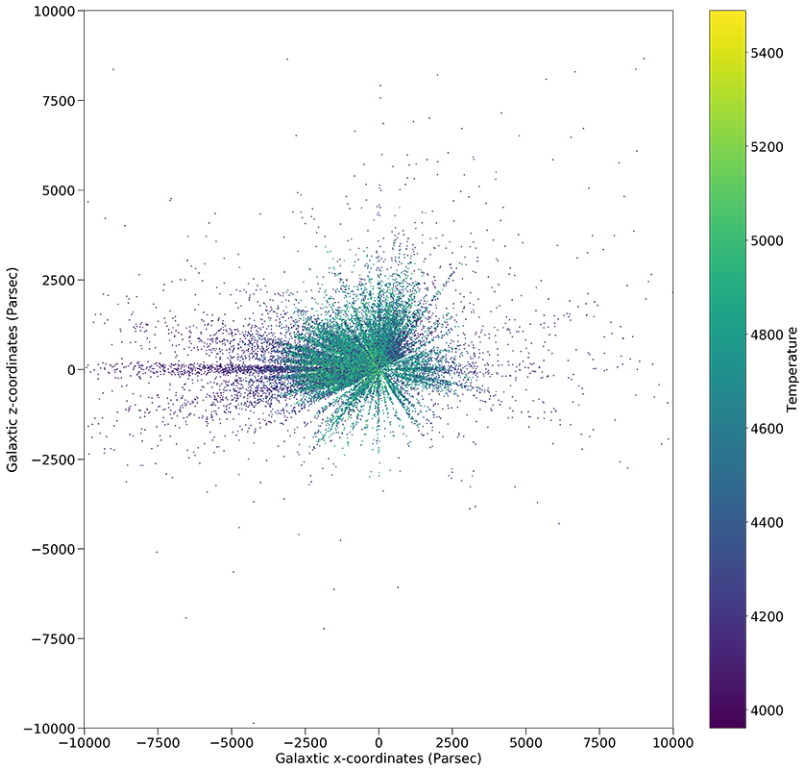
5.2.7 Milkyway via the Eye of Neural Network

Internal model identifier for the author: `astroNN_0224_run002`

Both the temperature and distance are the prediction from neural network. Combined with the observed coordinates and apparent magnitude, we can get a 3D map of stellar parameters via a neural network.

It seems like the neural network constantly overestimating the intrinsic brightness of low temperature stars, thats why it seems like low temperature stars dominated at distant.





- Uncertainty Analysis of Neural Nets with Variational Methods
- Galaxy10 Notebook
- neuralnets/vae_demo

- Variational AutoEncoder with simple 1D data demo
- Training neural net with DR14 APOGEE_Distances Value Added Catalogue using astroNN
- Gaia DR2 things

CHAPTER 6

APOGEE/Gaia/LAMOST Tools and Spectra Analysis using astroNN

6.1 Mini Tools for APOGEE data - astroNN.apogee

Note: astroNN only contains a limited amount of necessary tools. For a more comprehensive python tool to deal with APOGEE data, please refer to Jo Bovy's APOGEE tools

The APO Galactic Evolution Experiment 1 (APOGEE-1) employed high-resolution, high signal-to-noise infrared spectroscopy to penetrate the dust that obscures significant fractions of the disk and bulge of our Galaxy. APOGEE surveyed over 100,000 red giant stars across the full range of the Galactic bulge, bar, disk, and halo. APOGEE-1 generated precise radial velocities and detailed chemical abundances, providing unprecedented insights into the dynamical structure and chemical history of the Galaxy. In conjunction with the planet-finding surveys, Kepler and CoRoT, APOGEE unravels problems in fundamental astrophysics.

SDSS APOGEE: http://www.sdss.org/surveys/apogee/

6.1.1 Continuum Normalization of APOGEE Spectra

You can access the default astroNN continuum mask fro APOGEE spectra by

```
import os
import astroNN
import numpy as np

dr = 14

dir = os.path.join(os.path.dirname(astroNN.__path__[0]), 'astroNN', 'data', f'dr{dr}_
↪contmask.npy')
cont_mask = np.load(dir)
```

When you do continuum normalization using astroNN, you can just use con_mask=None to use default mask provided by Jo Bovy's APOGEE Tools. astroNN will use a SINGLE continuum pixel mask to normalize all spectra you provided. Moreover, astroNN will normalize the spectra by chips instead of normalize them all together.

astroNN.apogee.**apogee_continuum**(*spectra*, *spectra_err*, *cont_mask=None*, *deg=2*, *dr=None*, *bitmask=None*, *target_bit=None*, *mask_value=1.0*)

It is designed only for apogee spectra by fitting Chebyshev polynomials to the flux values in the continuum mask by chips. The resulting continuum will have the same shape as *fluxes*.

Parameters

- **spectra** (*ndarray*) – spectra
- **spectra_err** (*ndarray*) – spectra uncertainty, same shape as spectra
- **cont_mask** (*ndarray[bool]*) – continuum mask
- **deg** (*int*) – The degree of Chebyshev polynomial to use in each region, default is 2 which works the best so far
- **dr** (*int*) – apogee dr
- **bitmask** (*ndarray*) – bitmask array of the spectra, same shape as spectra
- **target_bit** (*Union(int, list[int], ndarray[int])*) – a list of bit to be masked
- **mask_value** (*Union(int, float)*) – if a pixel is determined to be a bad pixel, this value will be used to replace that pixel flux

Returns normalized spectra, normalized spectra uncertainty

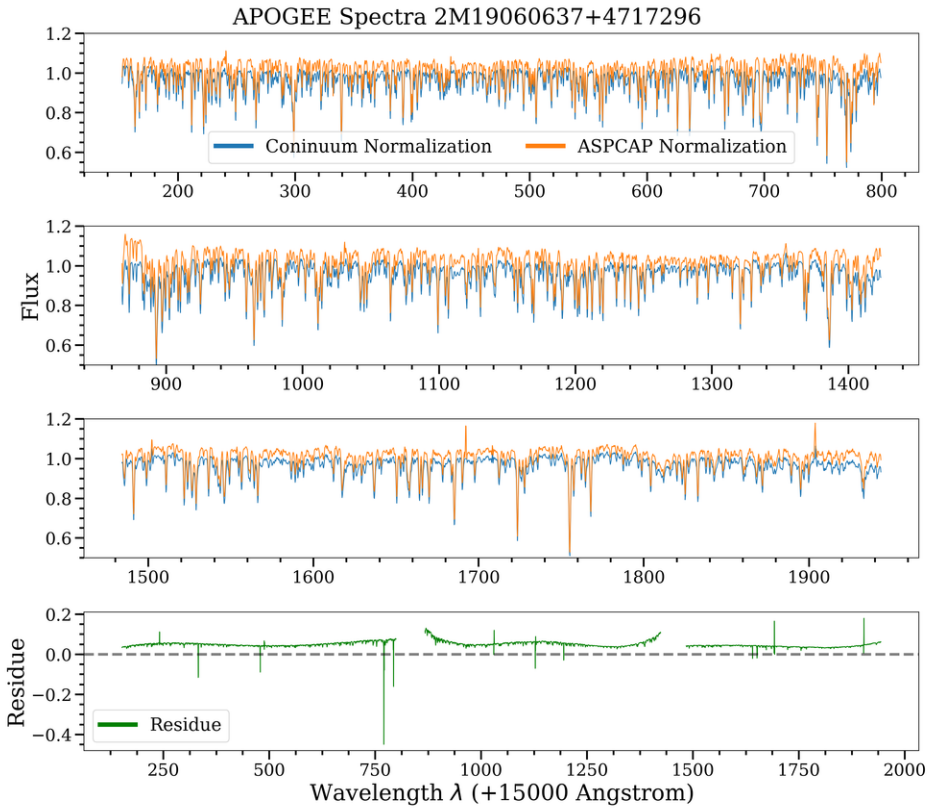
Return type ndarray, ndarray

History 2018-Mar-21 - Written - Henry Leung (University of Toronto)

```
from astroNN.apogee import apogee_continuum

# spectra_errs refers to the 1-sigma error array provided by APOGEE
# spectra can be multiple spectra at a time
norm_spec, norm_spec_err = apogee_continuum(apogee_spectra, spectra_errs, cont_
↪mask=None, deg=2, dr=14)

# If you deal with bitmask too and want to set some target bits to zero, you can add␣
↪additional arguement in apogee_continuum()
# You target_bit=[a list of number] or target_bit=None to use default target_bit
apogee_continuum(apogee_spectra, spectra_errs, cont_mask=None, deg=2, dr=14,␣
↪bitmask=apogee_bitmask, target_bit=None)
```

norm_spec refers to the normalized spectra while *norm_spec_err* refers to the normalized spectra error

Note: If you are planning to compile APOGEE dataset using astroNN, you can ignore this section as astroNN H5Compiler will load data from fits files directly and will take care everything.



You can use `continuum()` to normalize any spectra while `apogee_continuum()` is specifically designed for APOGEE spectra.

```
from astroNN.apogee import continuum

spec, spec_err = continuum(spectra, spectra_errs, cont_mask, deg=2)
```

6.1.2 Basics Tools related to APOGEE Spectra

Here are some basic tools to deal with APOGEE spectra

Retrieve Basic APOGEE Spectra Pixel Information

You can retrieve basic APOGEE spectra pixel information by

astroNN.apogee.**chips_pix_info**(*dr=None*)

To return chips info according to dr

Parameters **dr** (*Union(int, NoneType)*) – data release

Returns

The starting and ending pixels location of APOGEE camera chips in the original 8575 pixels spectra

- list[0] refers to the location where blue chips starts
- list[1] refers to the location where blue chips ends

- list[2] refers to the location where green chips starts
- list[3] refers to the location where blue chips end
- list[4] refers to the location where red chips starts
- list[5] refers to the location where red chips ends
- list[6] refers to the total number of pixels after deleting gap

Return type list

History

2017-Nov-27 - Written - Henry Leung (University of Toronto)
2017-Dec-16 - Updated - Henry Leung (University of Toronto)

```
from astroNN.apogee import chips_pix_info

info = chips_pix_info(dr=14)

# info[0] refers to the location where blue chips starts
# info[1] refers to the location where blue chips ends
# info[2] refers to the location where green chips starts
# info[3] refers to the location where blue chips end
# info[4] refers to the location where red chips starts
# info[5] refers to the location where red chips ends
# info[6] refers to the total number of pixels after deleting gap
```

APOGEE Spectra Wavelength Solution

astroNN.apogee.**wavelength_solution**(*dr=None*)

To return wavelegnth_solution, apStarWavegrid was provided by Jo Bovy's apogee tools (Toronto)

Parameters **dr** (*Union(int, NoneType)*) – data release

Returns

lambda_blue, lambda_green, lambda_red which are 3 wavelength solution array
- lambda_blue refers to the wavelength solution for each pixel in blue chips
- lambda_green refers to the wavelength solution for each pixel in green chips
- lambda_red refers to the wavelength solution for each pixel in red chips

Return type ndarray

History

2017-Nov-20 - Written - Henry Leung (University of Toronto)
2017-Dec-16 - Updated - Henry Leung (University of Toronto)

You can retrieve APOGEE spectra wavelength solution by

```
from astroNN.apogee import wavelength_solution

lambda_blue, lambda_green, lambda_red = wavelength_solution(dr=14)

# lambda_blue refers to the wavelength solution for each pixel in blue chips
```

(continues on next page)

(continued from previous page)

```
# lambda_green refers to the wavelength solution for each pixel in green chips
# lambda_red refers to the wavelength solution for each pixel in red chips
```

APOGEE Spectra Gap Delete

astroNN.apogee.**gap_delete**(*spectra*, *dr=None*)

To delete the gap between APOGEE CCDs from the original 8575 pixels spectra

Parameters

- **spectra** (*ndarray*) – The original 8575 pixels spectrum/spectra
- **dr** (*Union(int, NoneType)*) – data release

Returns Gap deleted spectrum/spectra

Return type ndarray

History

2017-Oct-26 - Written - Henry Leung (University of Toronto)
2017-Dec-16 - Updated - Henry Leung (University of Toronto)

You can delete the gap between raw spectra by

```
from astroNN.apogee import gap_delete

# original_spectra can be multiple spectra at a time
gap_deleted_spectra = gap_delete(original_spectra, dr=14)
```

Split APOGEE Spectra into Three Detectors

astroNN.apogee.**chips_split**(*spectra*, *dr=None*)

To split APOGEE spectra into RGB chips, will delete the gap if detected

Parameters

- **spectra** (*ndarray*) – APOGEE spectrum/spectra
- **dr** (*Union(int, NoneType)*) – data release

Returns 3 ndarrays which are spectra_blue, spectra_green, spectra_red

Return type ndarray

History

2017-Nov-20 - Written - Henry Leung (University of Toronto)
2017-Dec-17 - Updated - Henry Leung (University of Toronto)

You can split APOGEE spectra into three detectors by

```
from astroNN.apogee import chips_split

# original_spectra can be multiple spectra at a time
spectra_blue, spectra_green, spectra_red = chips_split(original_spectra, dr=14)
```

chips_split() will delete the gap between the detectors if you give raw APOGEE spectra. If you give gap deleted spectra, then the function will simply split the spectra into three.

APOGEE Bitmask to Boolean Array

You can turn a APOGEE PIXMASK bitmask array into a boolean array provided you have some target bit you want to mask

Bitmask: http://www.sdss.org/dr14/algorithms/bitmasks/#collapseAPOGEE_PIXMASK

astroNN.apogee.**bitmask_boolean**(*bitmask*, *target_bit*)

Turn bitmask to boolean with provided bitmask array and target bit to mask

Parameters

- **bitmask** (*ndarray*) – bitmask
- **target_bit** (*list[int]*) – target bit to mask

Returns boolean array, True for clean, False for masked

Return type ndarray[bool]

History 2018-Feb-03 - Written - Henry Leung (University of Toronto)

Example:

```
from astroNN.apogee import bitmask_boolean
import numpy as np

spectra_bitmask = np.array([2048, 128, 1024, 512, 16, 8192, 4096, 64, 2, 32, 256, 8,␣
↪4, 16896])
boolean_output = bitmask_boolean(spectra_bitmask, target_bit=[0,1,2,3,4,5,6,7,9,12])
print(boolean_output)
>>> array([[False, True, False, True, True, False, True, True, True, True, False,␣
↪True, True, True]])
```

Decompose APOGEE Bitmask into Constitute Bits

You can turn a APOGEE PIXMASK bit into its constitute bits

Bitmask: http://www.sdss.org/dr14/algorithms/bitmasks/#collapseAPOGEE_PIXMASK

astroNN.apogee.**bitmask_decompositor**(*bit*)

To decompose a bit from bitmask array to individual bit

Parameters **bit** (*int*) – bitmask

Returns boolean array, True for clean, False for masked

Return type ndarray[bool]

History 2018-Feb-03 - Written - Henry Leung (University of Toronto)

```
from astroNN.apogee import bitmask_decompositor

decomposed_bits = bitmask_decompositor(single_bitmask)
```

Example:

```
from astroNN.apogee import bitmask_decompositor

# Create a simulated bit number
# Lets say this pixel is marked as 0, 5, 13 and 14 bit
```

(continues on next page)

(continued from previous page)

```
bitmask = 2**0 + 2**5 + 2**13 + 2**14

decomposed_bits = bitmask_decompositor(bitmask)
# The function returns the set of original bits
>>> array([ 0,  5, 13, 14])
```

Retrieve ASPCAP Elements Window Mask

Original ASPCAP Elements Windows Mask: https://svn.sdss.org/public/repo/apogee/idlwrap/trunk/lib/ which is described in https://arxiv.org/abs/1502.04080

You can get ASPCAP elements window mask as a boolean array by providing an element name to this function,

`astroNN.apogee.`**`aspcap_mask`**(*elem*, *dr=None*)

To load ASPCAP elements window masks
DR14 Elements: `'C', 'CI', 'N', 'O', 'Na', 'Mg', 'Al', 'Si', 'P', 'S', 'K', 'Ca', 'TI', 'TiII', 'V', 'Cr', 'Mn', 'Fe', 'Co', 'Ni', 'Cu', 'Ge', 'Ce', 'Rb', 'Y', 'Nd'`

Parameters

- **elem** (*str*) – element name
- **dr** (*int*) – apogee dr

Returns mask

Return type ndarray[bool]

History 2018-Mar-24 - Written - Henry Leung (University of Toronto)

```
from astroNN.apogee import aspcap_mask

mask = aspcap_mask('Mg')  # for example you want to get ASPCAP Mg mask
```

6.1.3 APOGEE Data Downloader

astroNN APOGEE data downloader always act as functions that will return you the path of downloaded file(s), and download it if it does not exist locally. If the file cannot be found on server, astroNN will generally return `False` as the path.

General Way to Open Fits File

astropy.io.fits documentation: http://docs.astropy.org/en/stable/io/fits/

```
from astropy.io import fits

data = fits.open(local_path_to_file)
```

allstar file

Data Model: https://data.sdss.org/datamodel/files/APOGEE_REDUX/APRED_VERS/APSTAR_VERS/ASPCAP_VERS/RESULTS_VERS/allStar.html

astroNN.apogee.**allstar**(*dr=None*, *flag=None*)

Download the allStar file (catalog of ASPCAP stellar parameters and abundances from combined spectra)

Parameters

- **dr** (*int*) – APOGEE DR
- **flag** (*int*) – 0: normal, 1: force to re-download

Returns full file path and download in background if not found locally, False if cannot be found on server

Return type str

History 2017-Oct-09 - Written - Henry Leung (University of Toronto)

```
from astroNN.apogee import allstar

local_path_to_file = allstar(dr=14)
```

allvisit file

Data Model: https://data.sdss.org/datamodel/files/APOGEE_REDUX/APRED_VERS/APSTAR_VERS/ASPCAP_VERS/RESULTS_VERS/allVisit.html

astroNN.apogee.**allvisit**(*dr=None*, *flag=None*)

Download the allVisit file (catalog of properties from individual visit spectra)

Parameters

- **dr** (*int*) – APOGEE DR
- **flag** (*int*) – 0: normal, 1: force to re-download

Returns full file path and download in background if not found locally, False if cannot be found on server

Return type str

History 2017-Oct-11 - Written - Henry Leung (University of Toronto)

```
from astroNN.apogee import allvisit

local_path_to_file = allvisit(dr=14)
```

Combined Spectra (aspcapStar)

Data Model: https://data.sdss.org/datamodel/files/APOGEE_REDUX/APRED_VERS/APSTAR_VERS/ASPCAP_VERS/RESULTS_VERS/LOCATION_ID/aspcapStar.html

astroNN.apogee.**combined_spectra**(*dr=None*, *location=None*, *apogee=None*, *telescope=None*, *verbose=1*, *flag=None*)

Download the required combined spectra file a.k.a aspcapStar

Parameters

- **dr** (*int*) – APOGEE DR
- **location** (*int*) – Location ID [Optional]
- **apogee** (*str*) – Apogee ID
- **telescope** (*str*) – Telescope ID, for example 'apo25m' or 'lco25m'
- **flag** (*int*) – 0: normal, 1: force to re-download

Returns full file path and download in background if not found locally, False if cannot be found on server

Return type str

History

2017-Oct-15 - Written - Henry Leung (University of Toronto)
2018-Aug-31 - Updated - Henry Leung (University of Toronto)

```
from astroNN.apogee import combined_spectra

local_path_to_file = combined_spectra(dr=14, location=a_location_id, apogee=a_apogee_
↪id)
```

Visit Spectra (apStar)

Data Model: https://data.sdss.org/datamodel/files/APOGEE_REDUX/APRED_VERS/APSTAR_VERS/TELESCOPE/LOCATION_ID/apStar.html

astroNN.apogee.**visit_spectra**(*dr=None*, *location=None*, *apogee=None*, *telescope=None*, *verbose=1*, *flag=None*, *commission=False*)

Download the required individual spectra file a.k.a apStar

Parameters

- **dr** (*int*) – APOGEE DR
- **location** (*int*) – Location ID [Optional]
- **apogee** (*str*) – Apogee ID
- **telescope** (*str*) – Telescope ID, for example 'apo25m' or 'lco25m'
- **verbose** (*int*) – verbose
- **flag** (*int*) – 0: normal, 1: force to re-download
- **commission** (*bool*) – whether the spectra is taken during commissioning

Returns full file path and download in background if not found locally, False if cannot be found on server

Return type str

History

2017-Nov-11 - Written - Henry Leung (University of Toronto)
2018-Aug-31 - Updated - Henry Leung (University of Toronto)

```
from astroNN.apogee import visit_spectra

local_path_to_file = visit_spectra(dr=14, location=a_location_id, apogee=a_apogee_id)
```

Red Clumps of SDSS Value Added Catalogs

Introduction: http://www.sdss.org/dr14/data_access/value-added-catalogs/?vac_id=apogee-red-clump-rc-catalog

Data Model (DR14): https://data.sdss.org/datamodel/files/APOGEE_RC/cat/apogee-rc-DR14.html

astroNN.datasets.apogee_rc.**load_apogee_rc**(*dr=None*, *metric='distance'*, *extinction=True*)

Load apogee red clumps (absolute magnitude measurement)

Parameters

- **dr** (*int*) – Apogee DR
- **metric** (*string*) – which metric you want to get back
  - "absmag" for k-band absolute magnitude
  - "fakemag" for k-band fake magnitude
  - "distance" for distance in parsec
- **extinction** (*bool*) – Whether to take extinction into account, only affect when metric is NOT 'distance'

Returns numpy array of ra, dec, metrics_array

Return type ndarrays

History

2018-Jan-21 - Written - Henry Leung (University of Toronto)
2018-May-12 - Updated - Henry Leung (University of Toronto)

```
from astroNN.apogee import apogee_vac_rc

local_path_to_file = apogee_vac_rc(dr=14)
```

Or you can use *load_apogee_rc()* to load the data by

```
from astroNN.datasets import load_apogee_rc

# metric can be 'distance' for distance in parsec, 'absmag' for k-band absolute␣
↪magnitude
# 'fakemag' for astroNN's k-band fakemag scale
RA, DEC, metrics_array = load_apogee_rc(dr=14, metric='distance', extinction=True)  #␣
↪extinction only effective if not metric='distance'
```

APOKASC in the Kepler Fields

```
from astroNN.datasets.apokasc import apokasc_load

ra, dec, logg = apokasc_load()

# OR you want the gold and basic standard separately
gold_ra, gold_dec, gold_logg, basic_ra, basic_dec, basic_logg = apokasc_
↪load(combine=False)
```

APOGEE DR14-Based Distance Estimations

Introduction: http://www.sdss.org/dr14/data_access/value-added-catalogs/?vac_id=apogee-dr14-based-distance-estimations

Data Model (DR14): https://data.sdss.org/datamodel/files/APOGEE_DISTANCES/apogee_distances.html

astroNN.apogee.**apogee_distances**(*dr=None*, *flag=None*)

Download the Apogee Distances catalogue

Parameters

- **dr** (*int*) – Apogee DR
- **flag** (*int*) – Force to download if flag=1

Returns full file path

Return type str

History 2018-Jan-24 - Written - Henry Leung (University of Toronto)

```
from astroNN.apogee.downloader import apogee_distances

local_path_to_file = apogee_distances(dr=14)
```

astroNN.datasets.**load_apogee_distances**(*dr=None*, *metric='distance'*, *cuts=True*, *extinction=True*, *keepdims=False*)

Load apogee distances (absolute magnitude from stellar model)

Parameters

- **dr** (*int*) – Apogee DR
- **metric** (*string*) – which metric you want to get back
  - "absmag" for absolute magnitude
  - "fakemag" for fake magnitude
  - "distance" for distance in parsec
- **cuts** (*Union[boolean, float]*) – Whether to cut bad data (negative parallax and percentage error more than 20%), or a float to set the threshold
- **extinction** (*bool*) – Whether to take extinction into account, only affect when metric is NOT 'distance'
- **keepdims** (*boolean*) – Whether to preserve indices the same as APOGEE allstar DR14, no effect when cuts=False, set to -9999 for bad indices when cuts=True keepdims=True

Returns numpy array of ra, dec, metrics_array, metrics_err_array

Return type ndarrays

History 2018-Jan-25 - Written - Henry Leung (University of Toronto)

Or you can use *load_apogee_distances()* to load the data by

```
from astroNN.datasets import load_apogee_distances

# metric can be 'distance' for distance in parsec, 'absmag' for k-band absolute␣
↪magnitude
# 'fakemag' for astroNN's k-band fakemag scale
```

(continues on next page)

(continued from previous page)

```
# cuts=True to cut out those unknown values (-9999.) and measurement error > 20%
RA, DEC, metrics_array, metrics_err_array = load_apogee_distances(dr=14, metric=
↪'distance', cuts=True, keepdims=False)
```

Cannon's allstar

Introduction: https://data.sdss.org/datamodel/files/APOGEE_REDUX/APRED_VERS/APSTAR_VERS/ASPCAP_VERS/RESULTS_VERS/CANNON_VERS/cannonModel.html

Data Model (DR14): https://data.sdss.org/datamodel/files/APOGEE_REDUX/APRED_VERS/APSTAR_VERS/ASPCAP_VERS/RESULTS_VERS/CANNON_VERS/allStarCannon.html

astroNN.apogee.**allstarcannon**(*dr=None*, *flag=None*)

Download the allStarCannon file (catalog of Cannon stellar parameters and abundances from combined spectra)

Parameters

- **dr** (*int*) – APOGEE DR
- **flag** (*int*) – 0: normal, 1: force to re-download

Returns full file path and download in background if not found locally, False if cannot be found on server

Return type str

History 2017-Oct-24 - Written - Henry Leung (University of Toronto)

```
from astroNN.apogee import allstarcannon

local_path_to_file = allstarcannon(dr=14)
```

6.2 Mini Tools for LAMOST data - astroNN.lamost

This module is designed for dealing with LAMOST DR5.

LAMOST DR5 is not a public data release yet, this module only provides a limited amount of tools to deal with the spectra. If you do not have the data, astroNN will not provide any LAMOST DR5 data nor functions to download them.

LAMOST Data Policy: http://www.lamost.org/policies/data_policy.html

LAMOST DR5 Homepage: http://dr5.lamost.org/

LAMOST DR5 Data Model: http://dr5.lamost.org/doc/data-production-description

6.2.1 LAMOST Spectra Wavelength Solution

astroNN.lamost.**wavelength_solution**(*dr=None*)

To return wavelegnth_solution

Parameters **dr** (*Union(int, NoneType)*) – data release

Returns wavelength solution array

Return type ndarray

History 2018-Mar-15 - Written - Henry Leung (University of Toronto)

You can retrieve LAMOST spectra wavelength solution by

```
from astroNN.lamost import wavelength_solution

lambda_solution = wavelength_solution(dr=5)
```

6.2.2 Pseudo-Continuum Normalization of LAMOST Spectra

astroNN.lamost.**pseudo_continuum**(*flux*, *ivar*, *wavelength=None*, *L=50*, *dr=None*)

Pseudo-Continuum normalise a spectrum by dividing by a Gaussian-weighted smoothed spectrum.

Parameters

- **flux** (*ndarray*) – The observed flux array.
- **ivar** (*ndarray*) – The inverse variances of the fluxes.
- **wavelength** (*ndarray*) – An array of the wavelengths.
- **L** (*int*) – [optional] The width of the Gaussian in pixels.
- **dr** (*int*) – [optional] dara release

Returns Continuum normalized flux and flux uncerteinty

Return type ndarray

```
from astroNN.lamost import pseudo_continuum

# spectra_errs refers to the inverse variance array provided by LAMOST
# spectra can be multiple spectra at a time
norm_spec, norm_spec_err = pseudo_continuum(spectra, spectra_errs, dr=5)
```

6.2.3 Load LAMOST DR5 catalogue

astroNN.lamost.**load_allstar_dr5**()

Open LAMOST DR5 allstar

Returns fits file opened by astropy

Return type astropy.io.fits.hdu.hdulist.HDUList

History 2018-Jun-17 - Written - Henry Leung (University of Toronto)

```
from astroNN.lamost import load_allstar_dr5

fits_file = load_allstar_dr5()
fits_file[1].header  # print file header
```

6.3 Mini Tools for Gaia data - astroNN.gaia

Note: astroNN only contains a limited amount of necessary tools. For a more comprehensive python tool to deal with Gaia data, please refer to Jo Bovy's gaia_tools

The mission of the GAIA spacecraft is to create a dynamic, three-dimensional map of the Milky Way Galaxy by measuring the distances, positions and proper motion of stars. To do this, the spacecraft employs two telescopes, an imaging system, an instrument for measuring the brightness of stars, and a spectrograph. Launched in 2013, GAIA orbits the Sun at Lagrange point L2, 1.5 million kilometres from Earth. By the end of its five-year mission, GAIA will have mapped well over one billion stars—one percent of the Galactic stellar population.

ESA Gaia satellite: http://sci.esa.int/gaia/

6.3.1 Gaia Data Downloader

astroNN Gaia data downloader always act as functions that will return you the path of downloaded file(s), and download it if it does not exist locally. If the file cannot be found on server, astroNN will generally return `False` as the path.

Load Gaia DR2 - Apogee DR14 matches

astroNN.gaia.**gaiadr2_parallax**(*cuts=True*, *keepdims=False*, *offset=False*)

Load Gaia DR2 - APOGEE DR14 matches, indices corresponds to APOGEE allstar DR14 file

Parameters

- **cuts** (*Union[boolean, float]*) – Whether to cut bad data (negative parallax and percentage error more than 20%), or a float to set the threshold
- **keepdims** (*boolean*) – Whether to preserve indices the same as APOGEE allstar DR14, no effect when cuts=False, set to -9999 for bad indices when cuts=True keepdims=True
- **offset** (*Union[boolean, float, str]*) – Whether to correction Gaia DR2 zero point offset
  - False to assume no offset correction
  - True to assume 52.8-4.21(G-12.2)
  - "leungbovy2019" for leung & bovy 2019 offset correction
  - a float to assume a float offset globally

Returns numpy array of ra, dec, parallax, parallax_error

Return type ndarrays

History 2018-Apr-26 - Written - Henry Leung (University of Toronto)

```
from astroNN.gaia import gaiadr2_parallax

# To load Gaia DR2 - APOGEE DR14 matches, indices corresponds to APOGEE allstar DR14␣
↪file
ra, dec, parallax, parallax_error = gaiadr2_parallax(cuts=True, keepdims=False,␣
↪offset=False)
```

Gaia DR1 TGAS Downloader and Loader

astroNN.gaia.**tgas**(*flag=None*)

Get path to the Gaia TGAS DR1 files, download if files not found

Returns List of file path

Return type list

History 2017-Oct-13 - Written - Henry Leung (University of Toronto)

To download TGAS DR1, moreover TGAS is only available in DR1

```
from astroNN.gaia import tgas

# To download tgas dr1 to GAIA_TOOLS_DATA and it will return the list of path to
↪those files
files_paths = tgas()
```

To load Gaia TGAS

astroNN.gaia.**tgas_load**(*cuts=True*)

To load useful parameters from multiple TGAS DR1 files

Parameters **cuts** (*Union[boolean, 0.2]*) – Whether to cut bad data (negative parallax and percentage error more than 20%, or a custom cut percentage)

Returns Dictionary of parameters

Return type dict

History 2017-Dec-17 - Written - Henry Leung (University of Toronto)

```
from astroNN.gaia import tgas_load

# To load the tgas DR1 files and return a dictionary of ra(J2015), dec(J2015), pmra,
↪pmdec, parallax, parallax error, g-band mag
# cuts=True to cut bad data (negative parallax and percentage error more than 20%)
output = tgas_load(cuts=True)

# outout dictionary
output['ra']  # ra(J2015)
output['dec']  # dec(J2015)
output['pmra']  # proper motion in RA
output['pmdec']  # proper motion in DEC
output['parallax']  # parallax
output['parallax_err']  # parallax error
output['gmag']  # g-band mag
```

Gaia_source DR1 Downloader

No plan to support DR2 Gaia Source, please refers to Jo Bovy's https://github.com/jobovy/gaia_tools

```
from astroNN.gaia import gaia_source

# To download gaia_source DR1 to GAIA_TOOLS_DATA and it will return the list of path
↪to those files
files_paths = gaia_source(dr=1)
```

Anderson et al 2017 Improved Parallax from Data-driven Stars Model

Anderson2017 is described in here: https://arxiv.org/pdf/1706.05055

Please be advised starting from 26 April 2018, anderson2017 in astroNN is reduced to parallax cross matched with APOGEE DR14 only. If you see this message, anderson2017 in this astroNN version is reduced. Moreover, anderson2017 will be removed in the future

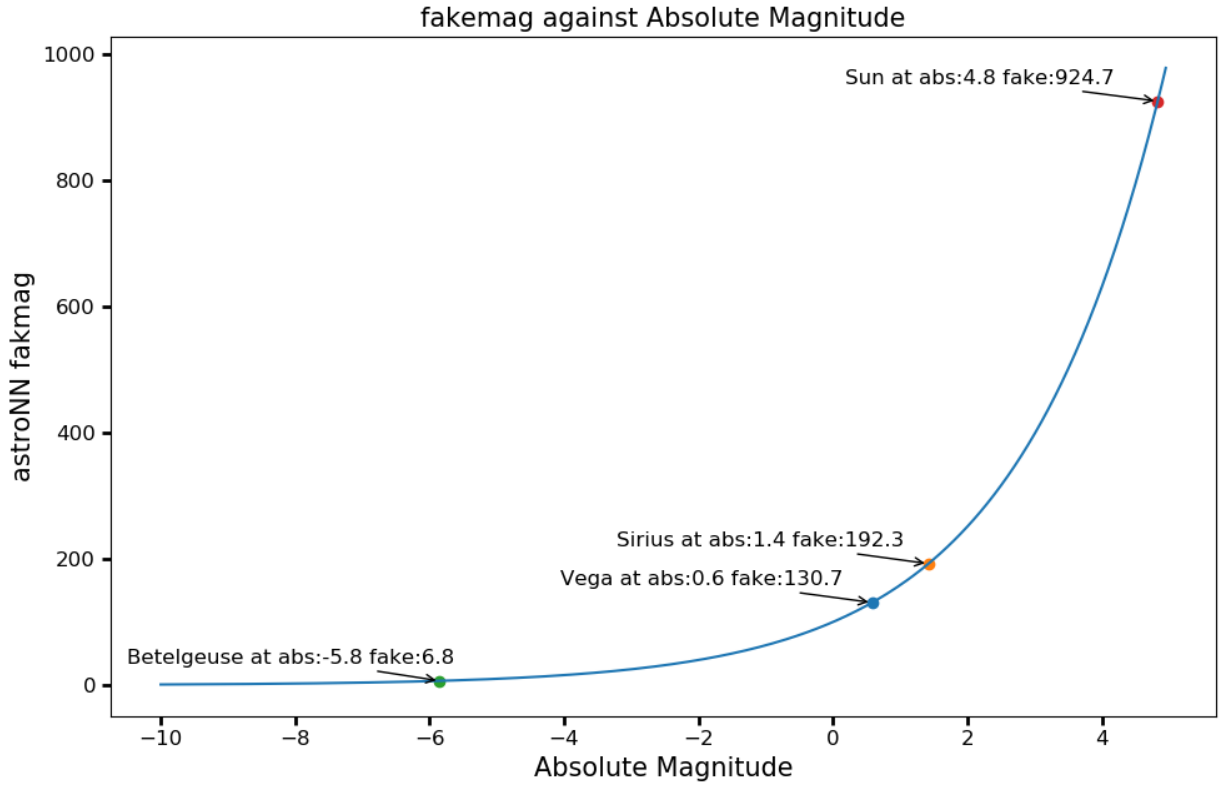
```
from astroNN.gaia import anderson_2017_parallax

# To load the improved parallax
# Both parallax and para_var is in mas
# cuts=True to cut bad data (negative parallax and percentage error more than 20%)
ra, dec, parallax, para_err = anderson_2017_parallax(cuts=True)
```

6.3.2 fakemag (dummy scale)

`fakemag` is an astroNN dummy scale primarily used to preserve the gaussian standard error from Gaia. astroNN always assume there is no error in apparent magnitude measurement.

$L_{\text{fakemag}} = \varpi 10^{\frac{1}{5}m_{\text{apparent}}} = 10^{\frac{1}{5}M_{\text{absolute}}+2}$, where ϖ is parallax in *mas*

You can get a sense of the fakemag scale from the following plot



6.3.3 Conversion Tools related to Astrometry and Magnitude

Some functions have input error argument, they are optional and if you provided error, the function will propagate error and have 2 returns (convened data, and converted propagated error), otherwise it will only has 1 return (converted data)

`astroNN.gaia.`**`mag_to_fakemag`**(*mag*, *parallax*, *parallax_err=None*)

To convert apparent magnitude to astroNN fakemag, Magic Number will be preserved

Parameters

- **mag** (*Union[float, ndarray]*) – apparent magnitude

- **parallax** (*Union[float, ndarray, astropy Quantity]*) – parallax (mas) or with astropy(can be distance with units) so astroNN will convert to appropriate units
- **parallax_err** (*Union[NoneType, float, ndarray, astropy Quantity]*) – parallax_error (mas) or with astropy so astroNN will convert to appropriate units

Returns astroNN fakemag, with addition (with additional return of propagated error if parallax_err is provided)

Return type Union[float, ndarray]

History 2017-Oct-14 - Written - Henry Leung (University of Toronto)

astroNN.gaia.**mag_to_absmag**(*mag*, *parallax*, *parallax_err=None*)
To convert apparent magnitude to absolute magnitude, Magic Number will be preserved

Parameters

- **mag** (*Union[float, ndarray]*) – apparent magnitude
- **parallax** (*Union[float, ndarray, astropy Quantity]*) – parallax (mas) or with astropy (can be distance with units) so astroNN will convert to appropriate units
- **parallax_err** (*Union[NoneType, float, ndarray, astropy Quantity]*) – parallax_error (mas) or with astropy so astroNN will convert to appropriate units

Returns absolute magnitude (with additional return of propagated error if parallax_err is provided)

Return type Union[float, ndarray]

History 2017-Oct-14 - Written - Henry Leung (University of Toronto)

astroNN.gaia.**absmag_to_pc**(*absmag*, *mag*)
To convert absolute magnitude to parsec, Magic Number will be preserved

Parameters

- **absmag** (*Union[float, ndarray]*) – absolute magnitude
- **mag** (*Union[float, ndarray]*) – apparent magnitude

Returns parsec

Return type astropy Quantity

History 2017-Nov-16 - Written - Henry Leung (University of Toronto)

astroNN.gaia.**fakemag_to_absmag**(*fakemag*)
To convert fakemag to absmag, Magic Number will be preserved

Parameters **fakemag** (*Union[float, ndarray]*) – eastroNN fakemag

Returns absolute magnitude

Return type Union[float, ndarray]

History 2018-Jan-31 - Written - Henry Leung (University of Toronto)

astroNN.gaia.**absmag_to_fakemag**(*absmag*)
To convert absmag to fakemag, Magic Number will be preserved

Parameters **absmag** (*Union[float, ndarray]*) – absolute magnitude

Returns astroNN fakemag

Return type Union[float, ndarray]

History 2018-Jan-31 - Written - Henry Leung (University of Toronto)

astroNN.gaia.**fakemag_to_pc**(*fakemag*, *mag*, *fakemag_err=None*)
To convert fakemag to parsec, Magic Number will be preserved

Parameters

- **fakemag** (*Union[float, ndarray]*) – astroNN fakemag
- **mag** (*Union[float, ndarray]*) – apparent magnitude
- **fakemag_err** (*Union[NoneType, float, ndarray]*) – Optional, fakemag_err

Returns array of pc with astropy Quantity (with additional return of propagated error if fakemag_err is provided)

Return type astropy Quantity

History 2018-Jan-31 - Written - Henry Leung (University of Toronto)

astroNN.gaia.**fakemag_to_parallax**(*fakemag*, *mag*, *fakemag_err=None*)
To convert fakemag to parallax, Magic Number will be preserved

Parameters

- **fakemag** (*Union[float, ndarray]*) – astroNN fakemag
- **mag** (*Union[float, ndarray]*) – apparent magnitude
- **fakemag_err** (*Union[NoneType, float, ndarray]*) – Optional, fakemag_err

Returns array of parallax in mas with astropy Quantity (with additional return of propagated error if fakemag_err is provided)

Return type astropy Quantity

History 2018-Aug-11 - Written - Henry Leung (University of Toronto)

astroNN.gaia.**fakemag_to_logsol**(*fakemag*, *band='K'*)

To convert fakemag to log10 solar luminosity, negative fakemag will be converted to MAGIC_NUMBER because of
fakemag cannot be negative in physical world

Parameters

- **fakemag** (*Union[float, ndarray]*) – astroNN fakemag
- **band** (*str(['U', 'B', 'V', 'R', 'I', 'J', 'H', 'K','u', 'g', 'r', 'i', 'z'])*) – band of your fakemag to use with

Returns log solar luminosity

Return type Union[float, ndarray]

History 2018-May-06 - Written - Henry Leung (University of Toronto)

astroNN.gaia.**absmag_to_logsol**(*absmag*, *band='K'*)
To convert absmag to log10 solar luminosity

Parameters

- **absmag** (*Union[float, ndarray]*) – absolute magnitude
- **band** (*str(['U', 'B', 'V', 'R', 'I', 'J', 'H', 'K','u', 'g', 'r', 'i', 'z'])*) – band of your absmag to use with

Returns log solar luminosity

Return type Union[float, ndarray]

History 2018-May-06 - Written - Henry Leung (University of Toronto)

astroNN.gaia.**logsol_to_fakemag**(*logsol*, *band='K'*)

To convert log solar luminosity to fakemag, negative fakemag will be converted to MAGIC_NUMBER because of fakemag
cannot be negative in physical world

Parameters

- **logsol** (*Union[float, ndarray]*) – log solar luminosity
- **band** (*str(['U', 'B', 'V', 'R', 'I', 'J', 'H', 'K','u', 'g', 'r', 'i', 'z'])*) – band of your fakemag to use with

Returns astroNN fakemag

Return type Union[float, ndarray]

History 2018-May-06 - Written - Henry Leung (University of Toronto)

astroNN.gaia.**logsol_to_absmag**(*logsol*, *band='K'*)

To convert log solar luminosity to absmag, negative fakemag will be converted to MAGIC_NUMBER because of fakemag
cannot be negative in physical world

Parameters

- **logsol** (*Union[float, ndarray]*) – log solar luminosity
- **band** (*str(['U', 'B', 'V', 'R', 'I', 'J', 'H', 'K','u', 'g', 'r', 'i', 'z'])*) – band of your absmag to use with

Returns absmag

Return type Union[float, ndarray]

History 2018-May-06 - Written - Henry Leung (University of Toronto)

astroNN.gaia.**fakemag_to_mag**(*fakemag*, *pc*, *pc_err=None*)
To convert apparent magnitude to astroNN fakemag, Magic Number will be preserved

Parameters

- **fakemag** (*Union[float, ndarray]*) – fakemag
- **pc** (*Union[float, ndarray, astropy Quantity]*) – parsec or with astropy (can be parallax with units) so astroNN will convert to appropriate units
- **pc_error** (*Union[NoneType, float, ndarray, astropy Quantity]*) – parsec uncertainty or with astropy so astroNN will convert to appropriate units

Returns astroNN fakemag, with addition (with additional return of propagated error if parallax_err is provided)

Return type Union[float, ndarray]

History 2018-Aug-1 - Written - Henry Leung (University of Toronto)

astroNN.gaia.**extinction_correction**(*mag*, *extinction*)

To correct magnitude with extinction, this function assumes extinction is at the same wavelength as the magnitude you have provided

Parameters

- **mag** (*Union[float, ndarray]*) – apparent magnitude
- **extinction** (*Union[float, ndarray]*) – extinction

Returns corrected magnitude

Return type Union[float, ndarray]

History 2018-May-13 - Written - Henry Leung (University of Toronto)

All of these functions preserve `magicnumber` in input(s) and can be imported by

```
from astroNN.gaia import ...
```

Preserving `magicnumber` means the indices which matched `magicnumber` in `config.ini` will be preserved, for example:

```
from astroNN.gaia import absmag_to_pc

print(absmag_to_pc([1., -9999.], [2., 1.]))
>>> <Quantity [15.84893192, -9999.] pc>

print(absmag_to_pc([1., -9999.], [-9999., 1.]))
>>> <Quantity [-9999., -9999.] pc>
```

Since some functions support astropy Quantity framework, you can convert between units easily. Example:

```
from astroNN.gaia import absmag_to_pc
from astropy import units as u
import numpy as np

# Example data of [Vega, Sirius, Betelgeuse]
absmag = np.array([0.582, 1.42, -5.85])
mag = np.array([0.03, -1.46, 0.5])
pc = absmag_to_pc(absmag, mag)  # The output - pc - carries astropy unit

# Convert to AU
distance_in_AU = pc.to(u.AU)

# Or convert to angle units by using astropy's equivalencies function
arcsec = pc.to(u.arcsec, equivalencies=u.parallax())
```

Since some functions support error propagation, lets say you are not familiar with `fakemag` and you want to know how standard error in `fakemag` propagate to `parsec`, you can for example

```
from astroNN.gaia import fakemag_to_pc

fakemag = 300
fakemag_err = 100
apparent_mag = 10

print(fakemag_to_pc(fakemag, apparent_mag, fakemag_err))
>>> (<Quantity 333.33333333 pc>, <Quantity 111.11111111 pc>)
```

6.3.4 Coordinates Matching between catalogs using Bovy's xmatch

Coordinates matching between catalogue can be done by *xmatch* which is just an exact copy from Jo Bovy's *gaia_tools*

Here is the documentation of xmatch from Jo Bovy

```
xmatch(cat1,cat2,maxdist=2, colRA1='RA',colDec1='DEC',epoch1=2000., colRA2='RA',
↪colDec2='DEC',epoch2=2000.,
       colpmRA2='pmra',colpmDec2='pmdec', swap=False)

cat1 = First catalog
cat2 = Second catalog
maxdist = (2) maximum distance in arcsec
colRA1 = ('RA') name of the tag in cat1 with the right ascension in degree in cat1␣
↪(assumed to be ICRS)
colDec1 = ('DEC') name of the tag in cat1 with the declination in degree in cat1␣
↪(assumed to be ICRS)
epoch1 = (2000.) epoch of the coordinates in cat1
colRA2 = ('RA') name of the tag in cat2 with the right ascension in degree in cat2␣
↪(assumed to be ICRS)
colDec2 = ('DEC') name of the tag in cat2 with the declination in degree in cat2␣
↪(assumed to be ICRS)
epoch2 = (2000.) epoch of the coordinates in cat2
colpmRA2 = ('pmra') name of the tag in cat2 with the proper motion in right ascension␣
↪in degree in cat2
           (assumed to be ICRS; includes cos(Dec)) [only used when epochs are␣
↪different]
colpmDec2 = ('pmdec') name of the tag in cat2 with the proper motion in declination␣
↪in degree in cat2
            (assumed to be ICRS) [only used when epochs are different]
swap = (False) if False, find closest matches in cat2 for each cat1 source, if False␣
↪do the opposite (important when one of the catalogs
```

Here is an example

```
from astroNN.datasets import xmatch
import numpy as np

# Some coordinates for cat1, J2000.
cat1_ra = np.array([36.,68.,105.,23.,96.,96.])
cat1_dec = np.array([72.,56.,54.,55.,88.,88.])

# Some coordinates for cat2, J2000.
cat2_ra = np.array([23.,56.,222.,96.,245.,68.])
cat2_dec = np.array([36.,68.,82.,88.,26.,56.])

# Using maxdist=2 arcsecond separation threshold, because its default, so not shown␣
↪here
# Using epoch1=2000. and epoch2=2000., because its default, so not shown here
# because both datasets are J2000., so no need to provide pmra and pmdec which␣
↪represent proper motion
idx_1, idx_2, sep = xmatch(cat1_ra, cat2_ra, colRA1=cat1_ra, colDec1=cat1_dec,␣
↪colRA2=cat2_ra, colDec2=cat2_dec, swap=False)

print(idx_1)
>>> [1 4 5]
print(idx_2)
>>> [5 3 3]
```

(continues on next page)

(continued from previous page)

```
print(cat1_ra[idx_1], cat2_ra[idx_2])
>>> [68. 96. 96.], [68. 96. 96.]

# What happens if we swap cat_1 and cat_2
idx_1, idx_2, sep = xmatch(cat2_ra, cat1_ra, colRA1=cat2_ra, colDec1=cat2_dec,␣
↪colRA2=cat1_ra, colDec2=cat1_dec, swap=False)

print(idx_1)
>>> [3 5]
print(idx_2)
>>> [4 1]
print(cat1_ra[idx_2], cat2_ra[idx_1])
>>> [96. 68.], [96. 68.]  # xmatch cant find all the match

# Because we have some repeated index in cat2, we should turn swap=True
idx_1, idx_2, sep = xmatch(cat2_ra, cat1_ra, colRA1=cat2_ra, colDec1=cat2_dec,␣
↪colRA2=cat1_ra, colDec2=cat1_dec, swap=True)

print(idx_1)
>>> [5 3 3]
print(idx_2)
>>> [1 4 5]
print(cat1_ra[idx_2], cat2_ra[idx_1])
>>> [68. 96. 96.], [68. 96. 96.]  # Yea, seems like xmatch found all the matched
```

6.4 Compiling and Loading APOGEE and Gaia Datasets - astroNN.datasets

6.4.1 Compiling APOGEE Dataset

```
from astroNN.datasets import H5Compiler

# To create a astroNN compiler instance
compiler = H5Compiler()

# To set the name of the resulting h5 datasets, here a 'test.h5' will be created
compiler.filename = 'test'

# To compile a .h5 datasets, use .compile() method
compiler.compile()
```

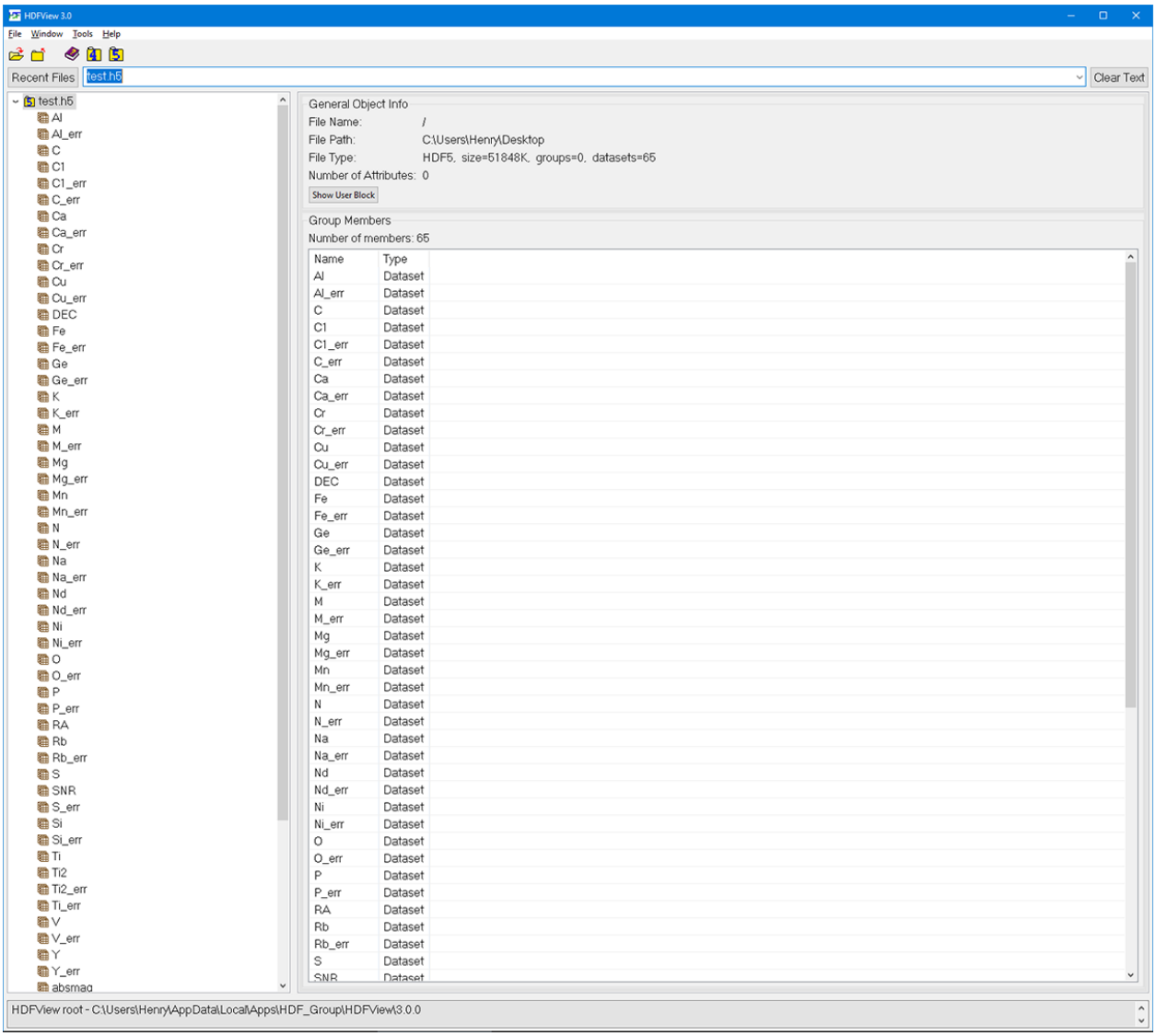
```
# Avaliable attributes of astroNN H5Compiler, set them before using H5Compiler.
↪compiler()

H5Compiler.apogee_dr  # APOGEE DR to use, Default is 14
H5Compiler.gaia_dr  # Gaia DR to use, Default is 1
H5Compiler.starflagcut = True  # True to filter out ASPCAP star flagged spectra
H5Compiler.aspcapflagcut = True  # True to filter out ASPCAP flagged spectra
H5Compiler.vscattercut = 1  # Upper bound of velocity scattering
H5Compiler.teff_high = 5500  # Upper bound of SNR
H5Compiler.teff_low = 4000  # Lower bound of SNR
H5Compiler.SNR_low = 200  # Lower bound of SNR
```

(continues on next page)

(continued from previous page)

```
H5Compiler.SNR_high = 99999  # Upper bound of SNR
H5Compiler.ironlow = -3  # Lower bound of SNR
H5Compiler.filename = None  # Filename of the resulting .h5 file
H5Compiler.spectra_only = False  # True to include spectra only without any aspcap␣
↪abundances
H5Compiler.cont_mask = None  # Continuum Mask, none to use default mask
H5Compiler.use_apogee = True  # Currently no effect
H5Compiler.use_esa_gaia = True  # True to use ESA Gaia parallax, **if use_esa_gaia is␣
↪True, ESA Gaia will has priority over Anderson 2017**
H5Compiler.use_anderson_2017 = False  # True to use Anderson et al 2017 parallax,␣
↪**if use_esa_gaia is True, ESA Gaia will has priority**
H5Compiler.err_info = True  # Whether to include error information in h5 dataset
H5Compiler.continuum = True  # True to do continuum normalization, False to use␣
↪aspcap normalized spectra
```

As a result, test.h5 will be created as shown below. you can use H5View to inspect the data



Note: For more detail on L. Anderson et al. (2017) improved parallax using data-driven stars model:

arXiv:1706.05055

6.4.2 Loading APOGEE Dataset

To load a compiled dataset, you can use

```
from astroNN.datasets import H5Loader

loader = H5Loader('datasets.h5')  # You should replace datasets.h5 with your real␣
↪filename
x, y = loader.load()
loader.load_err = True  # load error info too
x, y, x_err, y_err = loader.load()

# Lets say you want to load the corresponding SNR, apparent magnitude and coordinates␣
↪of the spectra loaded previously
snr = loader.load_entry('SNR')
kmag = loader.load_entry('Kmag')
ra = loader.load_entry('RA')
dec = loader.load_entry('DEC')
```

x will be an array of spectra [training data] and y will be an array of ASPCAP labels [training labels]

```
#Avaliable attributes of astroNN H5Loader, set them before H5Loader.load()
H5Loader.load_combined = True # Whether to load combined spectra or individual visits

#Target 'all' means ['teff', 'logg', 'M', 'alpha', 'C', 'C1', 'N', 'O', 'Na', 'Mg',
↪'Al', 'Si', 'P', 'S', 'K', 'Ca', 'Ti', 'Ti2', 'V', 'Cr',
#'Mn', 'Fe', 'Co', 'Ni', 'fakemag']
H5Loader.target = 'all'

# Whether to exclude all spectra contains -9999 in any ASPCAP abundances, By default,␣
↪astroNN can handle -9999 in training data
H5Loader.exclude9999 = False

# Whether to load error data
H5Loader.load_err = True

# True to load combined spectra, False to load individual visit (If there is any in␣
↪the h5 dataset you compiled)
# Training on combined spectra and test on individual spectra is recommended
H5Loader.load_combined = True
```

You can also use scikit-learn train_test_split to split x and y into training set and testing set.

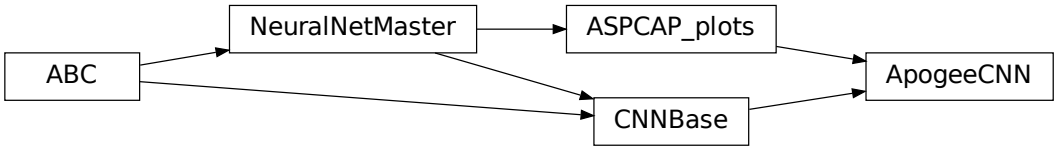
In case of APOGEE spectra, x_train and x_test are training and testing spectra. y_train and y_test are training and testing ASPCAP labels

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
```

6.5 APOGEE Spectra with Convolutional Neural Net - astroNN.models.ApogeeCNN

class astroNN.models.apogee_models.**ApogeeCNN**(*lr=0.005*)
Class for Convolutional Neural Network for stellar spectra analysis

History 2017-Dec-21 - Written - Henry Leung (University of Toronto)



Although in theory you can feed any 1D data to astroNN neural networks. This tutorial will only focus on spectra analysis.

```
from astroNN.models import ApogeeCNN
from astroNN.datasets import H5Loader

# Load the train data from dataset first, x_train is spectra and y_train will be␣
↪ASPCAP labels
loader = H5Loader('datasets.h5')
loader.load_err = False
x_train, y_train = loader.load()

# And then create an instance of Convolutional Neural Network class
cnn_net = ApogeeCNN()

# You don't have to specify the task because its 'regression' by default. But if you␣
↪are doing classification. you can set task='classification'
cnn_net.task = 'regression'

# Set max_epochs to 10 for a quick result. You should train more epochs normally
cnn_net.max_epochs = 10
cnn_net.train(x_train, y_train)
```

Here is a list of parameter you can set but you can also not set them to use default

```
ApogeeCNN.batch_size = 64
ApogeeCNN.initializer = 'he_normal'
ApogeeCNN.activation = 'relu'
ApogeeCNN.num_filters = [2, 4]
ApogeeCNN.filter_len = 8
ApogeeCNN.pool_length = 4
ApogeeCNN.num_hidden = [196, 96]
ApogeeCNN.max_epochs = 250
ApogeeCNN.lr = 0.005
ApogeeCNN.reduce_lr_epsilon = 0.00005
ApogeeCNN.reduce_lr_min = 0.0000000001
ApogeeCNN.reduce_lr_patience = 10
```

(continues on next page)

(continued from previous page)

```
ApogeeCNN.target = 'all'
ApogeeCNN.l2 = 1e-7
ApogeeCNN.input_norm_mode = 1
ApogeeCNN.labels_norm_mode = 2
```

Note: You can disable astroNN data normalization via `ApogeeCNN.input_norm_mode=0` as well as `ApogeeCNN.labels_norm_mode = 0` and do normalization yourself. But make sure you don't normalize labels with `MAGIC_NUMBER` (missing labels).

After the training, you can use *cnn_net* in this case and call test method to test the neural network on test data. Or you can load the folder by

```
from astroNN.models import load_folder
cnn_net = load_folder('astroNN_0101_run001')

# Load the test data from dataset, x_test is spectra and y_test will be ASPCAP labels
loader2 = H5Loader('datasets.h5')
loader2.load_combined = False
x_test, y_test = loader2.load()

pred = cnn_net.test(x_test)  # pred contains denormalized result aka. ASPCAP labels␣
↪prediction in this case
```

Since *astroNN.models.ApogeeCNN* does not have uncertainty analysis feature. You can plot aspcap label residue by supplying zeros arrays as error value. If you want model uncertainty/ risk estimation and propagated error, please use *astroNN.models.ApogeeBCNN*.

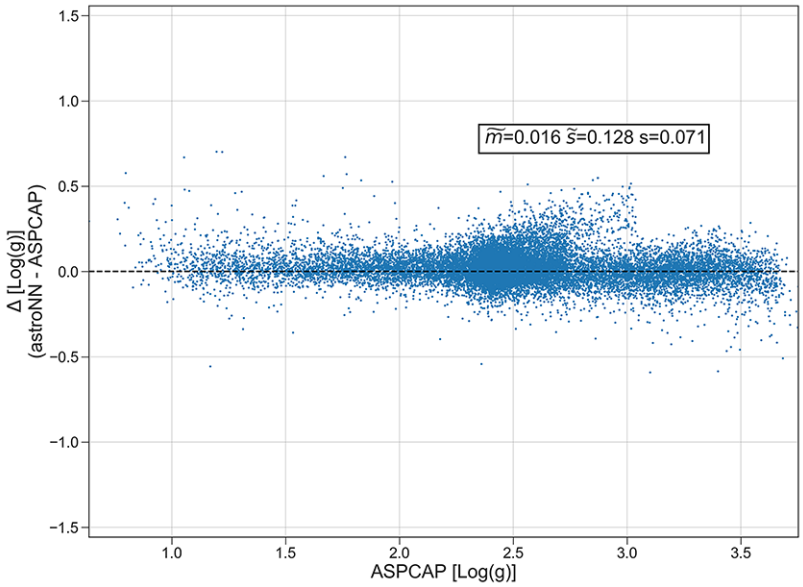
```
import numpy as np
cnn_net.aspcap_residue_plot(pred, y_test, np.zeros(y_test.shape))
```

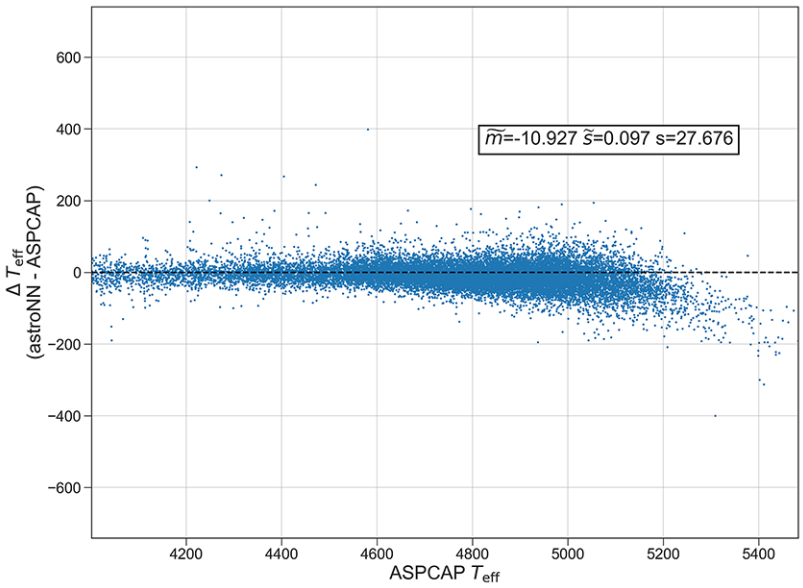
You can calculate jacobian which represents the output derivative to the input and see where those output is sensitive to in inputs.

```
# Calculate jacobian first
jacobian_array = cnn_net.jacobian(x_test, mean_output=True)

# Plot the graphs
cnn_net.jacobian_aspcap(jacobian=jacobian_array, dr=14)
```

Note: You can access to Keras model method like model.predict via (in the above tutorial) cnn_net.keras_model (Example: cnn_net.keras_model.predict())

6.5.1 Example Plots using aspcap_residue_plot





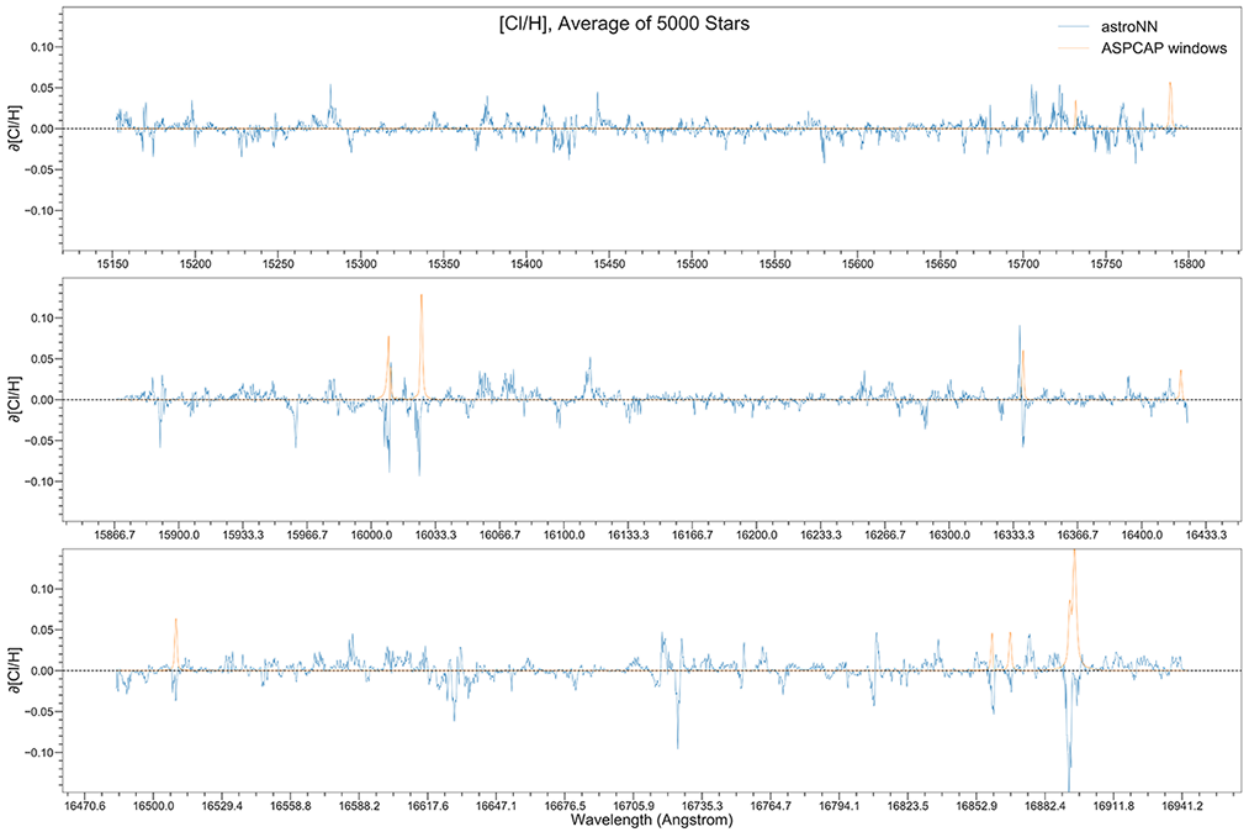
6.5.2 ASPCAP labels prediction using CNN vs The Cannon 2

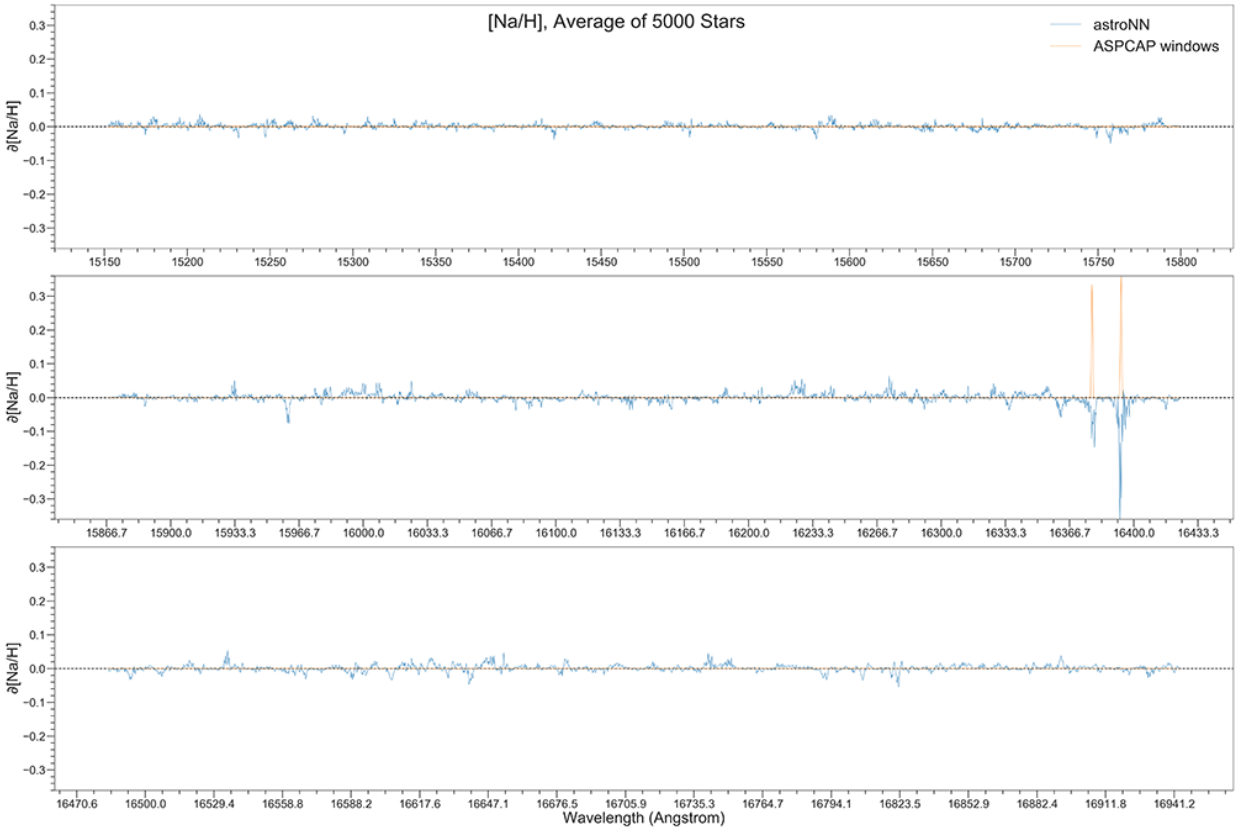
Warning: Please refer to Bayesian Neural Network for the most updated result: http://astronn.readthedocs.io/en/latest/neuralnets/apogee_bcnn.html

1. astroNN: astroNN with test data (cuts, 100<SNR<200), pooling 17 pixels, and 256 neurons in the first dense layer, otherwise the hyperparameters are the same as (3)
2. Cannon: Cannon with test data same as astroNN (cuts, 100<SNR<200)
3. astroNN: astroNN with test data (cuts, 100<SNR<200), pooling 4 pixels, and only 128 neurons in the first dense layer, otherwise the hyperparameters are the same as (1)

Labels	Median	MAD of residues / Training data SD	MAD of residues
Al	-0.022 / -0.044 / -0.031	0.340 / 0.483 / 0.328	0.077 / 0.109 / 0.074
Alpha/M	0.000 / -0.013 / -0.001	0.338 / 0.433 / 0.286	0.022 / 0.028 / 0.019
C	-0.004 / -0.024 / 0.003	0.217 / 0.280 / 0.188	0.057 / 0.066 / 0.044
Ca	-0.008 / -0.003 / -0.008	0.176 / 0.383 / 0.157	0.036 / 0.078 / 0.032
C1	-0.009 / - / 0.006	0.297 / - / 0.277	0.060 / - / 0.056
Cr	-0.002 / -0.010 / 0.002	0.204 / 0.238 / 0.194	0.048 / 0.056 / 0.045
Fe	0.003 / 0.010 / -0.004	0.115 / 0.153 / 0.098	0.027 / 0.036 / 0.023
Log(g)	0.003 / 0.021 / 0.016	0.150 / 0.146 / 0.128	0.083 / 0.081 / 0.071
M	-0.000 / 0.001 / -0.002	0.108 / 0.149 / 0.088	0.025 / 0.035 / 0.020
Mg	-0.002 / -0.001 / -0.002	0.190 / 0.201 / 0.164	0.037 / 0.039 / 0.032
Mn	-0.010 / -0.028 / -0.013	0.143 / 0.162 / 0.128	0.043 / 0.049 / 0.038
N	-0.002 / -0.012 / -0.005	0.177 / 0.309 / 0.163	0.053 / 0.093 / 0.049
Na	-0.010 / -0.082 / -0.016	0.519 / 0.696 / 0.510	0.160 / 0.215 / 0.157
Ni	-0.001 / -0.001 / -0.003	0.158 / 0.240 / 0.141	0.035 / 0.053 / 0.031
O	-0.009 / -0.040 / -0.014	0.262 / 0.401 / 0.253	0.049 / 0.075 / 0.047
P	-0.016 / -0.070 / -0.009	0.541 / 0.672 / 0.469	0.143 / 0.177 / 0.124
S	0.006 / 0.039 / 0.006	0.410 / 0.439 / 0.401	0.072 / 0.077 / 0.070
Si	-0.002 / 0.005 / -0.006	0.200 / 0.218 / 0.180	0.037 / 0.040 / 0.033
Teff	-19.218 / -28.737 / -10.927	0.132 / 0.154 / 0.097	37.548 / 44.078 / 27.676
Ti	-0.013 / -0.040 / -0.018	0.209 / 0.586 / 0.192	0.051 / 0.144 / 0.047
Ti2	0.003 / - / -0.004	0.826 / - / 0.682	0.144 / - / 0.119
V	-0.002 / -0.130 / 0.000	0.340 / 0.651 / 0.329	0.095 / 0.182 / 0.092

6.5.3 Example Plots using jacobian



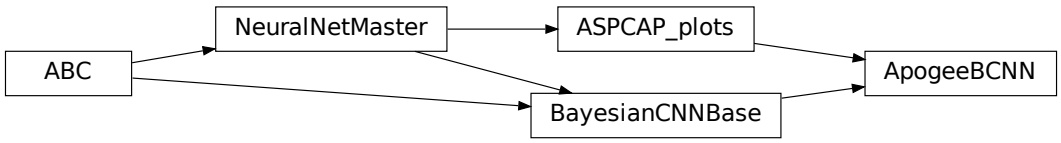


6.6 APOGEE Spectra with Bayesian Neural Net - astroNN.models.ApogeeBCNN

class astroNN.models.apogee_models.**ApogeeBCNN**(*lr=0.0005*, *dropout_rate=0.3*)

Class for Bayesian convolutional neural network for stellar spectra analysis

History 2017-Dec-21 - Written - Henry Leung (University of Toronto)



Although in theory you can feed any 1D data to astroNN neural networks. This tutorial will only focus on spectra analysis.

```
from astroNN.models import ApogeeBCNN
from astroNN.datasets import H5Loader
```

(continues on next page)

(continued from previous page)

```
# Load the train data from dataset first, x_train is spectra and y_train will be␣
↪ASPCAP labels
loader = H5Loader('datasets.h5')
loader.load_combined = True
loader.load_err = True
x_train, y_train, x_err, y_err = loader.load()

# And then create an instance of Bayesian Convolutional Neural Network class
bcnn_net = ApogeeBCNN()

# You don't have to specify the task because its 'regression' by default. But if you␣
↪are doing classification. you can set task='classification'
bcnn_net.task = 'regression'

# Set max_epochs to 10 for a quick result. You should train more epochs normally,␣
↪especially with dropout
bcnn_net.max_epochs = 10
bcnn_net.train(x_train, y_train, x_err, y_err)
```

Here is a list of parameter you can set but you can also not set them to use default

```
ApogeeBCNN.batch_size = 64
ApogeeBCNN.initializer = 'he_normal'
ApogeeBCNN.activation = 'relu'
ApogeeBCNN.num_filters = [2, 4]
ApogeeBCNN.filter_len = 8
ApogeeBCNN.pool_length = 4
ApogeeBCNN.num_hidden = [196, 96]
ApogeeBCNN.max_epochs = 100
ApogeeBCNN.lr = 0.005
ApogeeBCNN.reduce_lr_epsilon = 0.00005
ApogeeBCNN.reduce_lr_min = 0.0000000001
ApogeeBCNN.reduce_lr_patience = 10
ApogeeBCNN.target = 'all'
ApogeeBCNN.l2 = 5e-9
ApogeeBCNN.dropout_rate = 0.2
ApogeeBCNN.length_scale = 0.1  # prior length scale
ApogeeBCNN.input_norm_mode = 3
ApogeeBCNN.labels_norm_mode = 2
```

Note: You can disable astroNN data normalization via `ApogeeBCNN.input_norm_mode=0` as well as `ApogeeBCNN.labels_norm_mode=0` and do normalization yourself. But make sure you don't normalize labels with MAGIC_NUMBER (missing labels).

After the training, you can use *bcnn_net* in this case and call test method to test the neural network on test data. Or you can load the folder by

```
from astroNN.models import load_folder
bcnn_net = load_folder('astroNN_0101_run001')

# Load the test data from dataset, x_test is spectra and y_test will be ASPCAP labels
loader2 = H5Loader('datasets.h5')
loader2.load_combined = False
```

(continues on next page)

(continued from previous page)

```
loader2.load_err = False
x_test, y_test = loader2.load()

# pred contains denormalized result aka. ASPCAP labels prediction in this case
# pred_std is a list of uncertainty
# pred_std['total'] is the total uncertainty (standard derivation) which is the sum␣
↪of all the uncertainty
# pred_std['predictive'] is the predictive uncertainty predicted by bayesian neural␣
↪net
# pred_std['model'] is the model uncertainty from dropout variational inference
pred, pred_std = bcnn_net.test(x_test)
```

Since *astroNN.models.ApogeeBCNN* uses Bayesian deep learning which provides uncertainty analysis features. If you want quick testing/prototyping, please use *astroNN.models.ApogeeCNN*. You can plot aspcap label residue by

```
bcnn_net.aspcap_residue_plot(pred, y_test, pred_std['total'])
```

You can calculate jacobian which represents the output derivative to the input and see where those output is sensitive to in inputs.

```
# Calculate jacobian first
jacobian_array = bcnn_net.jacobian(x_test, mean_output=True)

# Plot the graphs
bcnn_net.jacobian_aspcap(jacobian=jacobian_array, dr=14)
```

Note: You can access to Keras model method like model.predict via (in the above tutorial) bcnn_net.keras_model (Example: bcnn_net.keras_model.predict())

6.6.1 ASPCAP Labels Prediction

Internal model identifier for the author: `astroNN_0321_run002`

Training set (30067 spectra + separate 3340 validation spectra): Starflag=0 and ASPCAPflag=0, 4000<Teff<5500, 200<SNR

Testing set (97723 spectra): Individual Visit of the training spectra, median SNR is around SNR~100

Using *astroNN.models.ApogeeBCNN* with default hyperparameter

Ground Truth is ASPCAP labels.

	Median of residue	astropy mad_std of residue
Al	-0.003	0.042
Alpha	0.000	0.013
C	0.003	0.032
C1	0.005	0.037
Ca	0.002	0.022
Co	-0.005	0.071
Cr	-0.001	0.031
fakemag	3.314	16.727
Fe	0.001	0.016
K	-0.001	0.032
Log(g)	0.002	0.048
M	0.003	0.015
Mg	0.001	0.021
Mn	0.003	0.025
N	-0.002	0.037
Na	-0.006	0.103
Ni	0.000	0.021
O	0.004	0.027
P	0.005	0.086
S	0.006	0.043
Si	0.001	0.022
Teff	0.841	23.574
Ti	0.002	0.032
Ti2	-0.009	0.089
V	-0.002	0.059

Median Absolute Error of prediction at three different low SNR level.

	SNR ~ 20	SNR ~ 40	SNR ~ 60
Al	0.122 dex	0.069 dex	0.046 dex
Alpha	0.024 dex	0.017 dex	0.014 dex
C	0.088 dex	0.051 dex	0.037 dex
C1	0.084 dex	0.054 dex	0.041 dex
Ca	0.069 dex	0.039 dex	0.029 dex
Co	0.132 dex	0.104 dex	0.085 dex
Cr	0.082 dex	0.049 dex	0.037 dex
fakemag	Not Calculated	Not Calculated	Not Calculated
Fe	0.070 dex	0.035 dex	0.024 dex
K	0.091 dex	0.050 dex	0.037 dex
Log(g)	0.152 dex	0.085 dex	0.059 dex
M	0.067 dex	0.033 dex	0.023 dex
Mg	0.080 dex	0.039 dex	0.026 dex
Mn	0.089 dex	0.050 dex	0.037 dex
N	0.118 dex	0.067 dex	0.046 dex
Na	0.119 dex	0.110 dex	0.099 dex
Ni	0.076 dex	0.039 dex	0.027 dex
O	0.076 dex	0.046 dex	0.037 dex
P	0.106 dex	0.082 dex	0.077 dex
S	0.072 dex	0.052 dex	0.041 dex
Si	0.076 dex	0.042 dex	0.024 dex
Teff	74.542 K	41.955 K	29.271 K
Ti	0.080 dex	0.049 dex	0.037 dex
Ti2	0.124 dex	0.099 dex	0.092 dex
V	0.119 dex	0.080 dex	0.064 dex

6.6.2 ASPCAP Labels Prediction with >50% corrupted labels

Internal model identifier for the author: `astroNN_0224_run004`

Setting is the same as above, but manually corrupt more labels to ensure the modified loss function is working fine

52.5% of the total training labels is corrupted to -9999 (4.6% of the total labels are -9999. from ASPCAP), while testing set is unchanged

	Median of residue	astropy mad_std of residue
Al	0.003	0.047
Alpha	0.000	0.015
C	0.005	0.037
C1	0.003	0.042
Ca	0.002	0.025
Co	0.001	0.076
Cr	0.000	0.033
fakemag	-0.020	5.766
Fe	0.001	0.020
K	0.001	0.035
Log(g)	-0.002	0.064
M	0.002	0.019
Mg	0.003	0.025
Mn	0.003	0.030
N	0.001	0.043
Na	-0.004	0.106
Ni	0.001	0.025
O	0.004	0.031
P	0.004	0.091
S	0.006	0.045
Si	0.001	0.026
Teff	-0.405	31.222
Ti	0.003	0.035
Ti2	-0.012	0.092
V	0.002	0.063

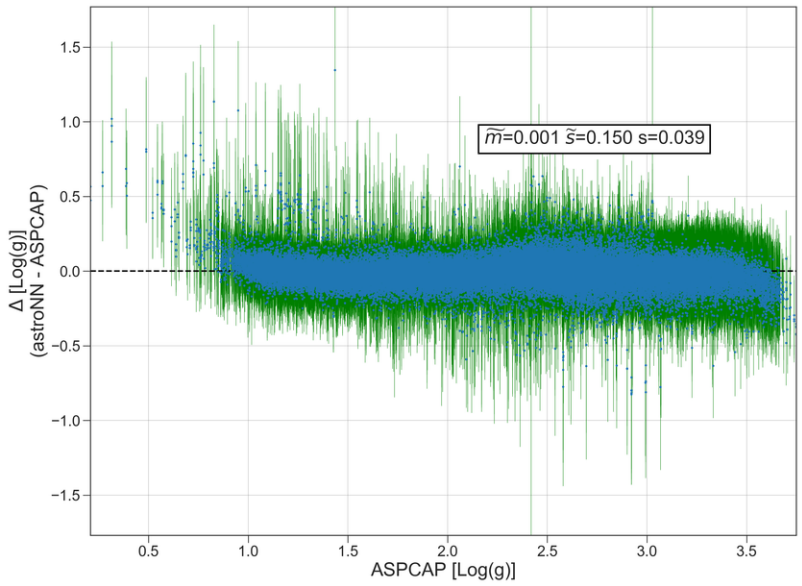
6.6.3 ASPCAP Labels Prediction with limited amount of data

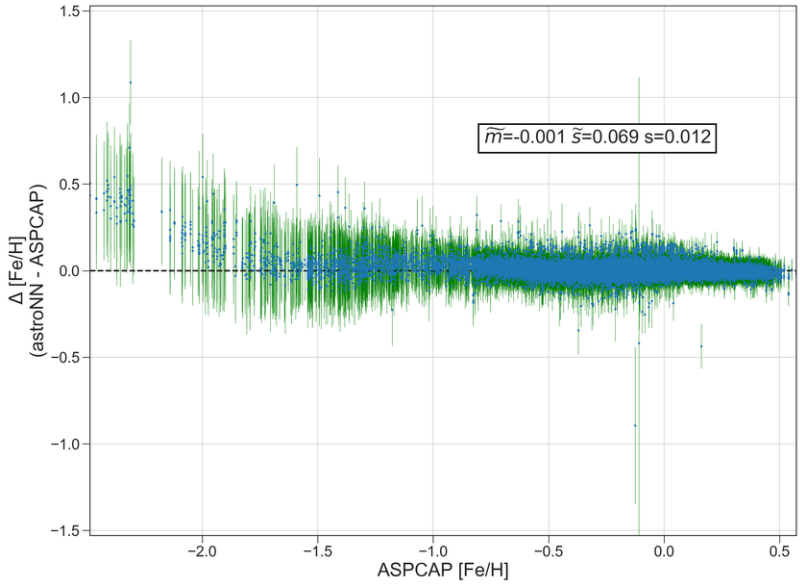
Internal model identifier for the author: `astroNN_0401_run001`

Setting is the same including the neural network, but the number of training data is limited to 5000 (4500 of them is for training, 500 validation), validation set is completely separated. Testing set is the same without any limitation.

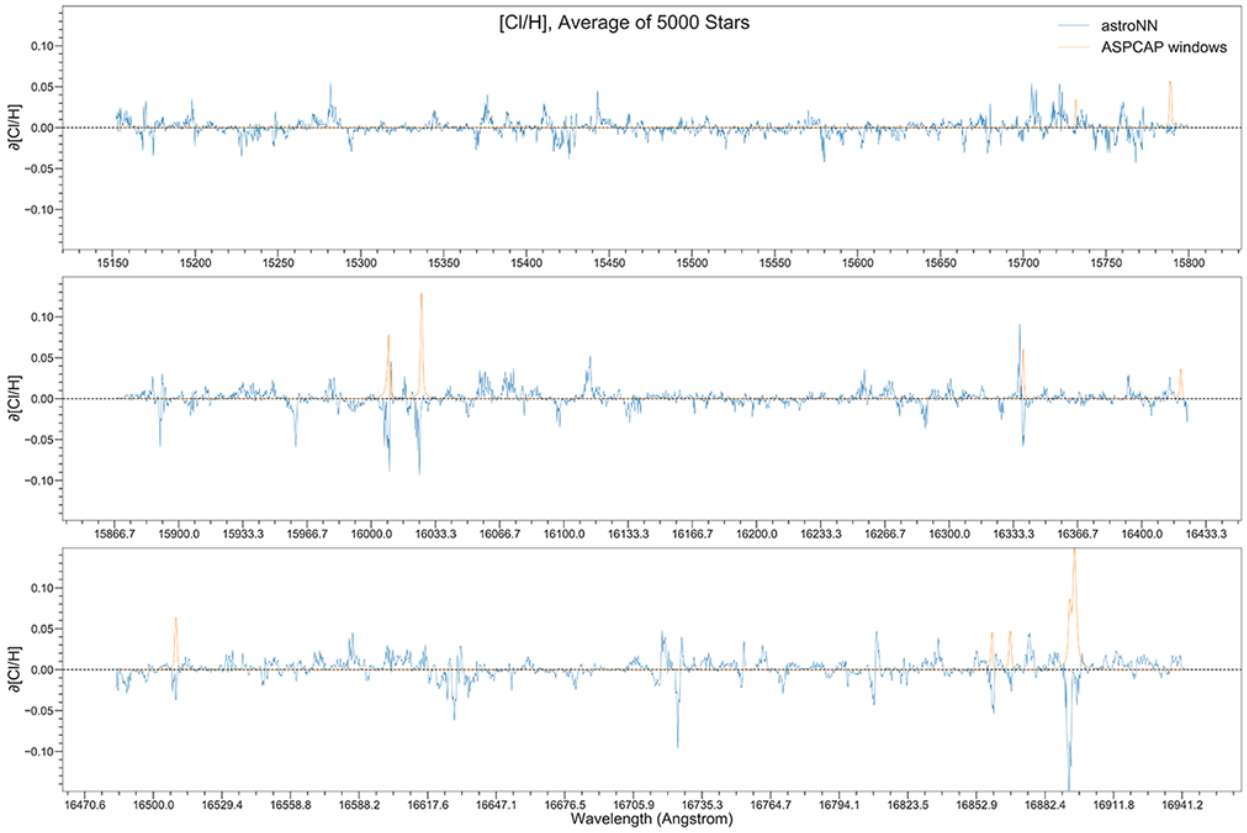
	Median of residue	astropy mad_std of residue
Al	-0.002	0.051
Alpha	0.001	0.017
C	-0.002	0.040
C1	-0.003	0.046
Ca	-0.003	0.027
Co	-0.006	0.080
Cr	0.000	0.036
fakemag	18.798	30.687
Fe	-0.004	0.022
K	-0.003	0.038
Log(g)	-0.005	0.064
M	-0.004	0.020
Mg	-0.002	0.026
Mn	-0.002	0.033
N	-0.003	0.053
Na	-0.026	0.121
Ni	-0.003	0.026
O	-0.003	0.033
P	0.001	0.097
S	-0.003	0.047
Si	-0.003	0.028
Teff	-1.348	33.202
Ti	-0.004	0.037
Ti2	-0.017	0.097
V	-0.005	0.065

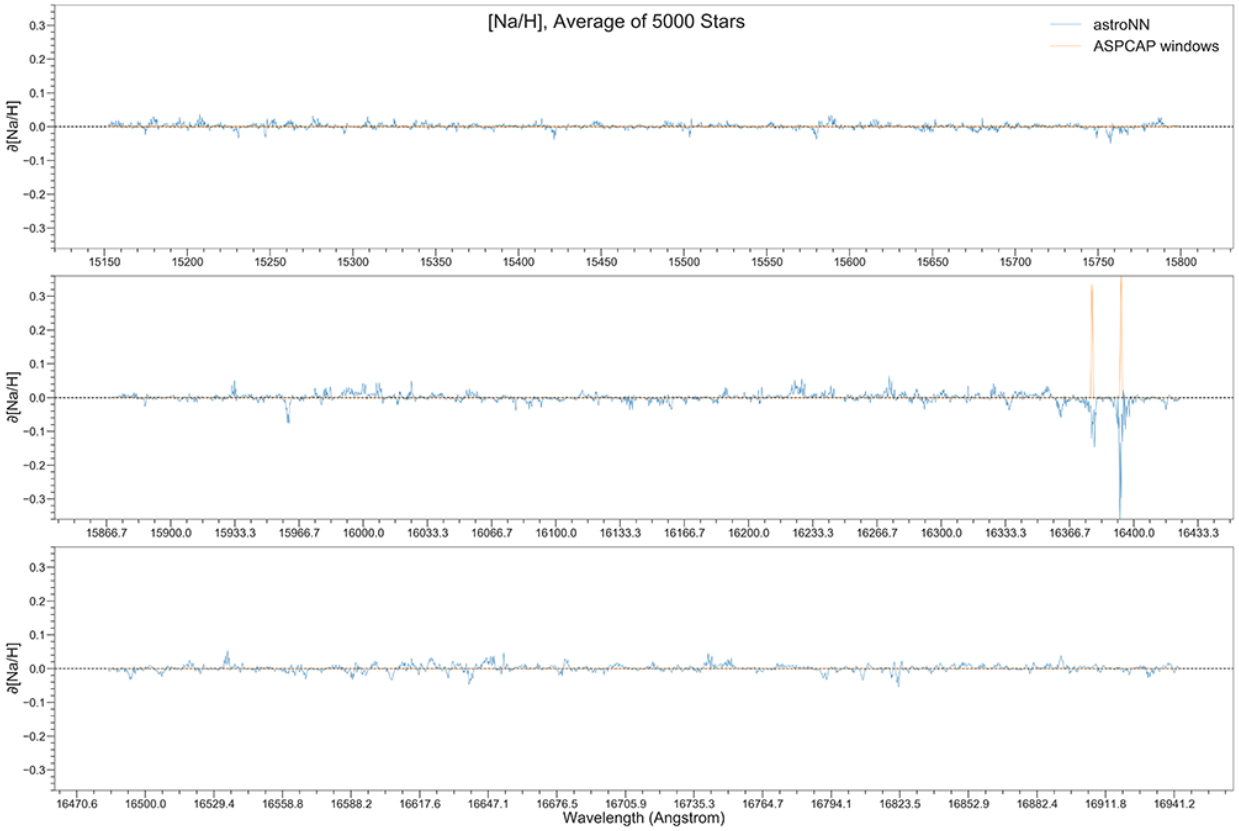
6.6.4 Example Plots using aspcap_residue_plot





6.6.5 Example Plots using jacobian





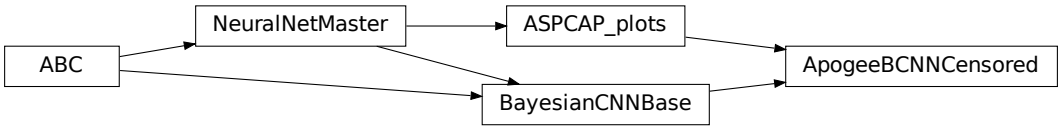
6.7 APOGEE Spectra with Censored Bayesian NN - astroNN.models.ApogeeBCNNCensored

class astroNN.models.apogee_models.**ApogeeBCNNCensored**(*lr=0.0005*, *dropout_rate=0.3*)

Class for Bayesian censored convolutional neural network for stellar spectra analysis [specifically APOGEE DR14 spectra only]

Described in the paper: https://ui.adsabs.harvard.edu/#abs/2018arXiv180804428L/abstract

History 2018-May-27 - Written - Henry Leung (University of Toronto)



ApogeeBCNNCensored can only be used with Apogee spectra with 7,514 pixels

```
from astroNN.models import ApogeeBCNNCensored
from astroNN.datasets import H5Loader

# Load the train data from dataset first, x_train is spectra and y_train will be␣
↪ASPCAP labels
loader = H5Loader('datasets.h5')
loader.load_combined = True
loader.load_err = False
loader.target = ['teff', 'logg', 'C', 'C1', 'N', 'O', 'Na', 'Mg', 'Al', 'Si', 'P', 'S
↪', 'K',
                 'Ca', 'Ti', 'Ti2', 'V', 'Cr', 'Mn', 'Fe','Co', 'Ni']
x_train, y_train, x_err, y_err = loader.load()

# And then create an instance of Apogee Censored Bayesian Convolutional Neural␣
↪Network class
bcnncensored_net = ApogeeBCNNCensored()

# Set max_epochs to 10 for a quick result. You should train more epochs normally,␣
↪especially with dropout
bcnncensored_net.max_epochs = 10
bcnncensored_net.train(x_train, y_train, x_err, y_err)
```

Here is a list of parameter you can set but you can also not set them to use default

```
ApogeeBCNNCensored.batch_size = 64
ApogeeBCNNCensored.initializer = 'he_normal'
ApogeeBCNNCensored.activation = 'relu'
ApogeeBCNNCensored.num_filters = [2, 4]
ApogeeBCNNCensored.filter_len = 8
ApogeeBCNNCensored.pool_length = 4
# number of neurone for [old_bcnn_1, old_bcnn_2, aspcap_1, aspcap_2, hidden]
ApogeeBCNNCensored.num_hidden = [128, 64, 32, 8, 2]
ApogeeBCNNCensored.max_epochs = 50
ApogeeBCNNCensored.lr = 0.005
ApogeeBCNNCensored.reduce_lr_epsilon = 0.00005
ApogeeBCNNCensored.reduce_lr_min = 0.0000000001
ApogeeBCNNCensored.reduce_lr_patience = 10
ApogeeBCNNCensored.target = 'all'
ApogeeBCNNCensored.l2 = 5e-9
ApogeeBCNNCensored.dropout_rate = 0.2
ApogeeBCNNCensored.length_scale = 0.1  # prior length scale
ApogeeBCNNCensored.input_norm_mode = 3
ApogeeBCNNCensored.labels_norm_mode = 2
```

Note: You can disable astroNN data normalization via `ApogeeBCNNCensored.input_norm_mode=0` as well as `ApogeeBCNNCensored.labels_norm_mode=0` and do normalization yourself. But make sure you don't normalize labels with MAGIC_NUMBER (missing labels).

After the training, you can use *bcnncensored_net* in this case and call test method to test the neural network on test data. Or you can load the folder by

```
from astroNN.models import load_folder
bcnncensored_net = load_folder('astroNN_0101_run001')

# Load the test data from dataset, x_test is spectra and y_test will be ASPCAP labels
```

(continues on next page)

(continued from previous page)

```
loader2 = H5Loader('datasets.h5')
loader2.load_combined = False
loader2.load_err = False
loader2.target = ['teff', 'logg', 'C', 'C1', 'N', 'O', 'Na', 'Mg', 'Al', 'Si', 'P', 'S
↪', 'K',
                  'Ca', 'Ti', 'Ti2', 'V', 'Cr', 'Mn', 'Fe','Co', 'Ni']
x_test, y_test = loader2.load()

# pred contains denormalized result aka. ASPCAP labels prediction in this case
# pred_std is a list of uncertainty
# pred_std['total'] is the total uncertainty (standard derivation) which is the sum␣
↪of all the uncertainty
# pred_std['predictive'] is the predictive uncertainty predicted by bayesian neural␣
↪net
# pred_std['model'] is the model uncertainty from dropout variational inference
pred, pred_std = bcnncensored_net.test(x_test)
```

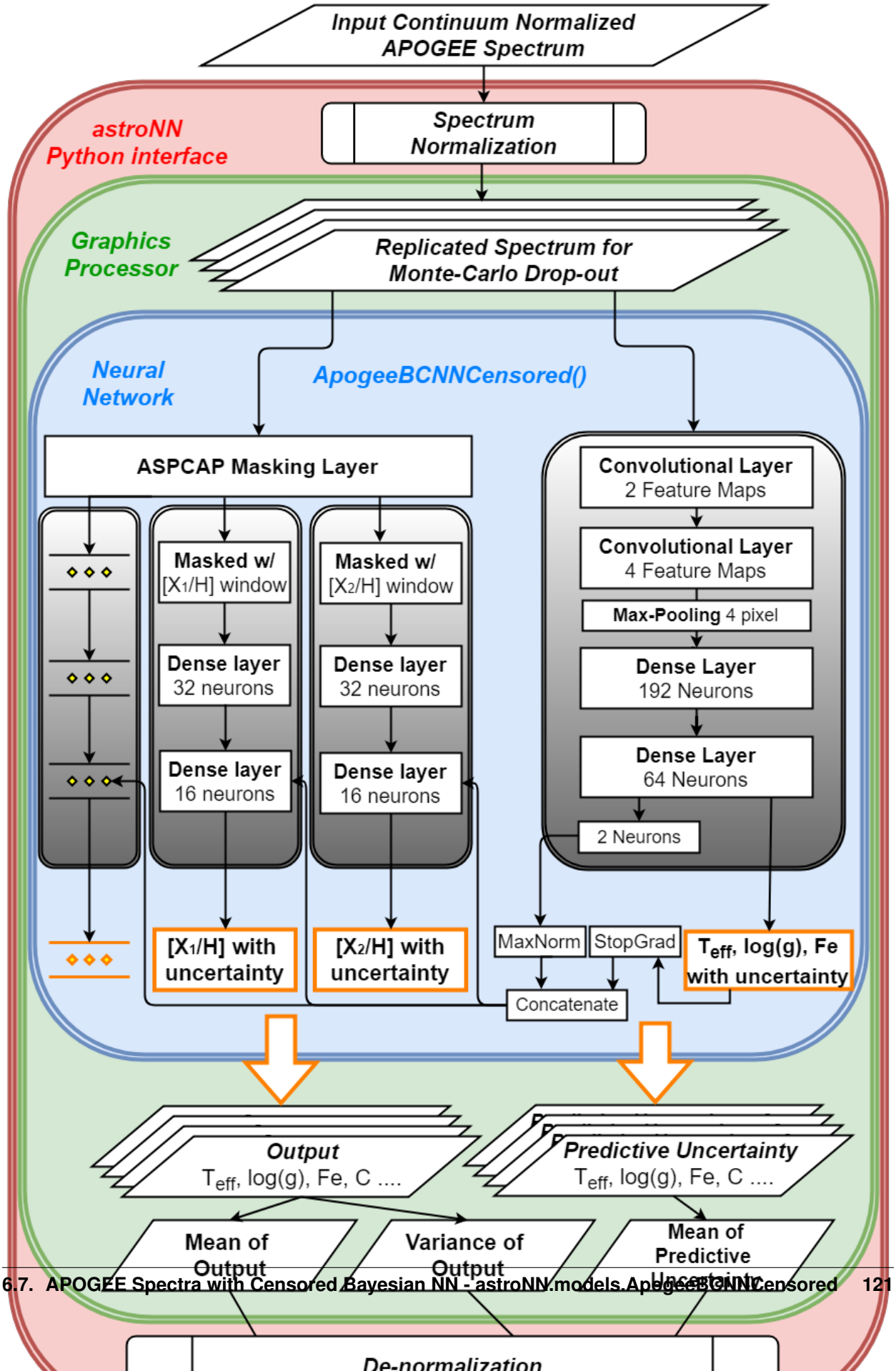
```
bcnncensored_net.aspcap_residue_plot(pred, y_test, pred_std['total'])
```

You can calculate jacobian which represents the output derivative to the input and see where those output is sensitive to in inputs.

```
# Calculate jacobian first
jacobian_array = bcnncensored_net.jacobian(x_test, mean_output=True)

# Plot the graphs
bcnncensored_net.jacobian_aspcap(jacobian=jacobian_array, dr=14)
```

6.7.1 Architecture

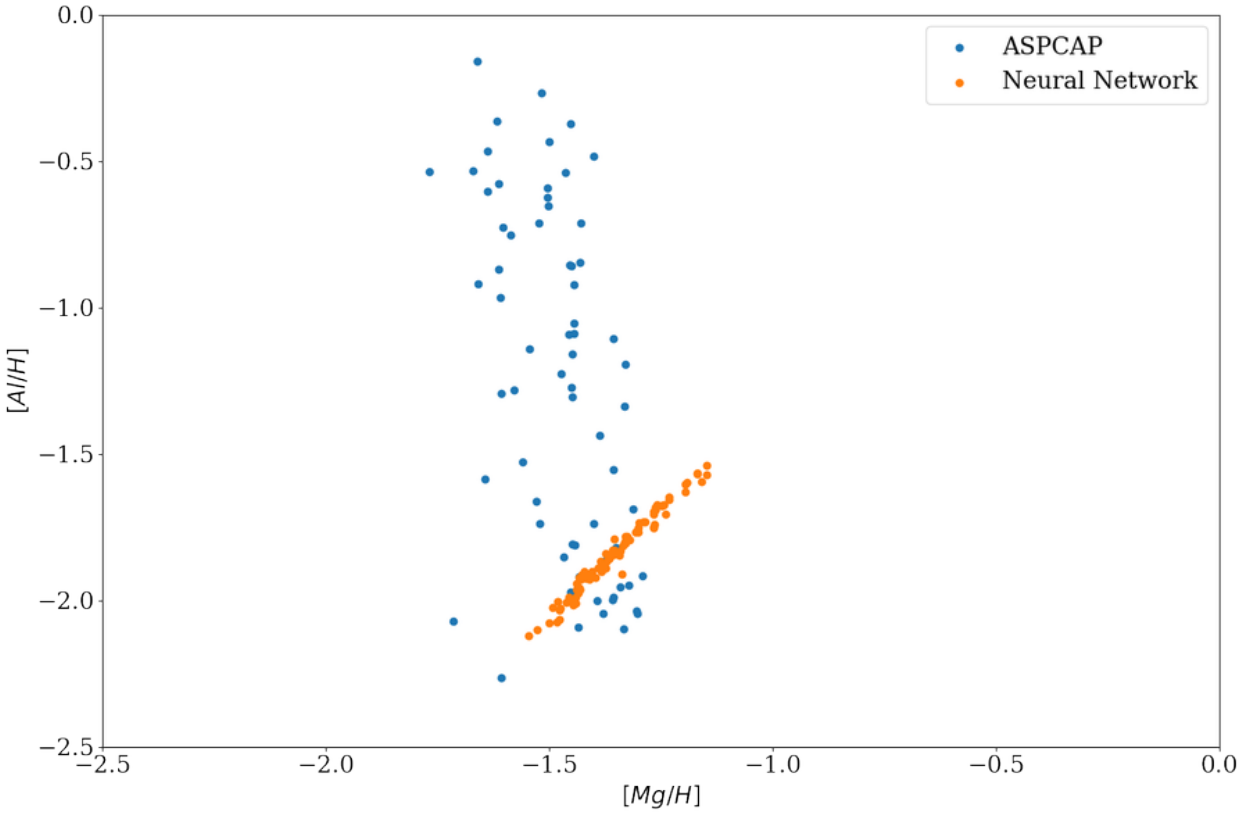
The architecture of this neural network is as follow.


Input Continuum Normalized APOGEE Spectrum
astroNN Python interface
Spectrum Normalization
Graphics Processor
Replicated Spectrum for Monte-Carlo Drop-out
Neural Network
ApogeeBCNNCensored()
ASPCAP Masking Layer
Masked w/ [X1/H] window
Masked w/ [X2/H] window
Dense layer 32 neurons
Dense layer 32 neurons
Dense layer 16 neurons
Dense layer 16 neurons
Convolutional Layer 2 Feature Maps
Convolutional Layer 4 Feature Maps
Max-Pooling 4 pixel
Dense Layer 192 Neurons
Dense Layer 64 Neurons
2 Neurons
[X1/H] with uncertainty
[X2/H] with uncertainty
MaxNorm
StopGrad
Teff, log(g), Fe with uncertainty
Concatenate
Output
Teff, log(g), Fe, C ....
Predictive Uncertainty
Teff, log(g), Fe, C ....
Mean of Output
Variance of Output
Mean of Predictive Uncertainty
De-normalization

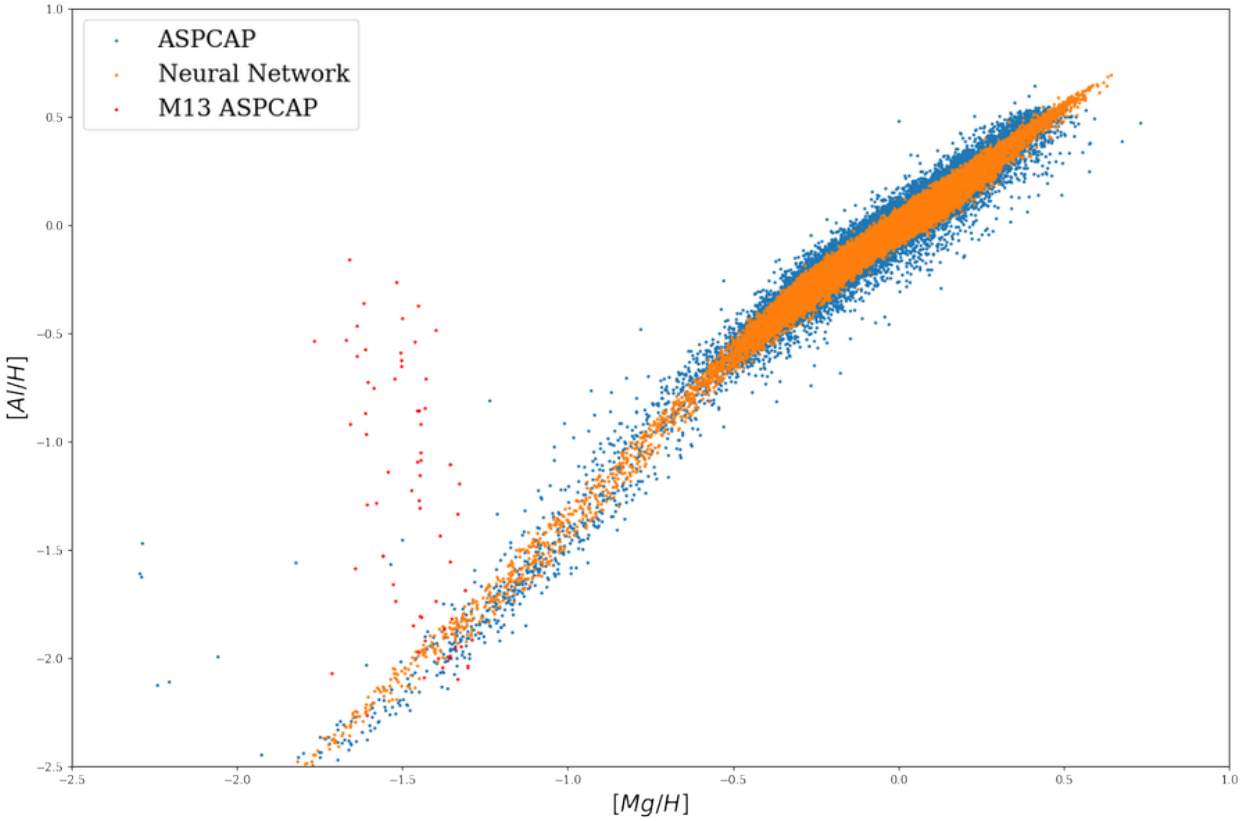
6.7.2 Why Censored NN for APOGEE Spectra analysis?

Internal model identifier for the author: `astroNN_0529_run010`

It caught our attention that *ApogeeBCNN* neural network found no spread in $[Al/H]$ in $M13$ globular cluster (Literature of showing a spread in $[Al/H]$: https://arxiv.org/pdf/1501.05127.pdf) and it may imply a problem in *ApogeeBCNN* that it found strongly correlation between elements but not actually measuring individually.

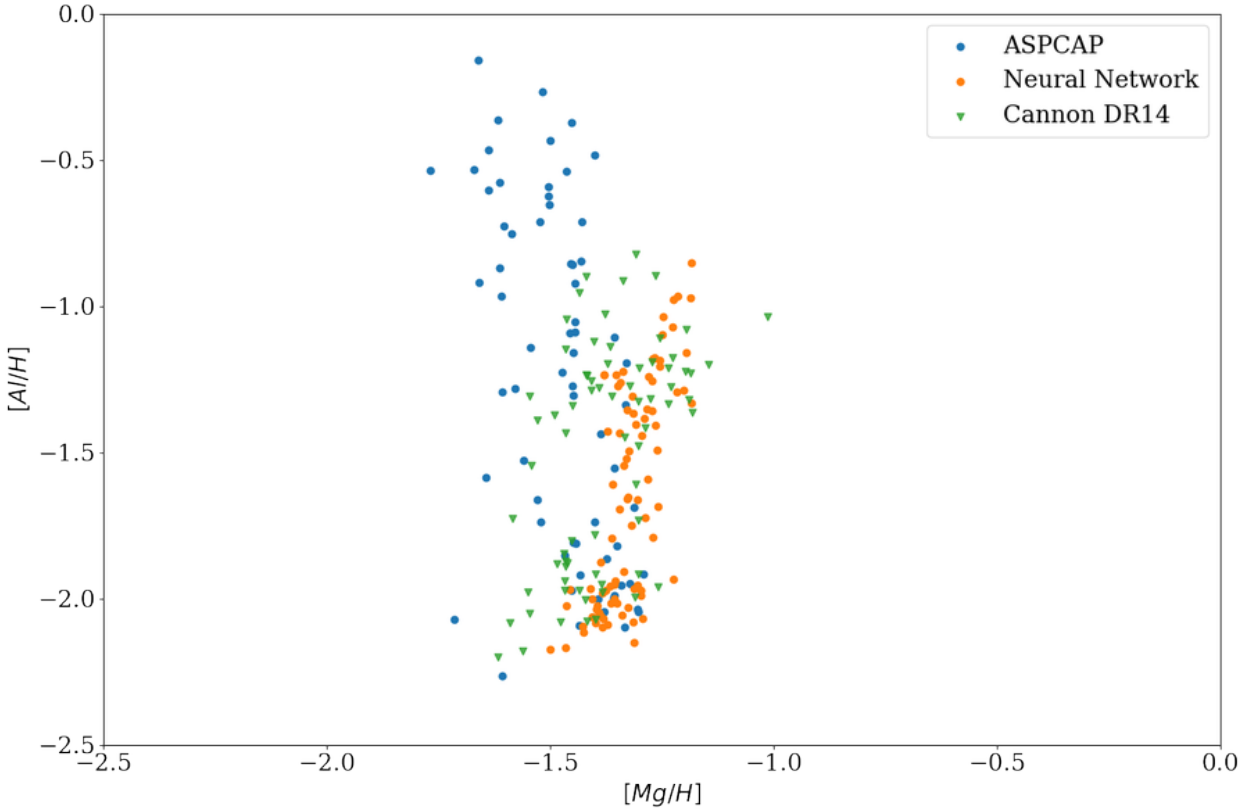


It becomes clear when we plot the training set $[Al/H]$ vs $[Mg/H]$ as follow, $[Al/H]$ and $[Mg/H]$ are strongly correlated and *ApogeeBCNN* is just measuring $[Al/H]$ as some kind of $[Mg/H]$ and fooled in $M13$ because $M13$ has a spread in $[Al/H]$ but not $[Mg/H]$, in other word, the region in $[Mg/H, Al/H]$ parameter space of $M13$ is not covered by training set.

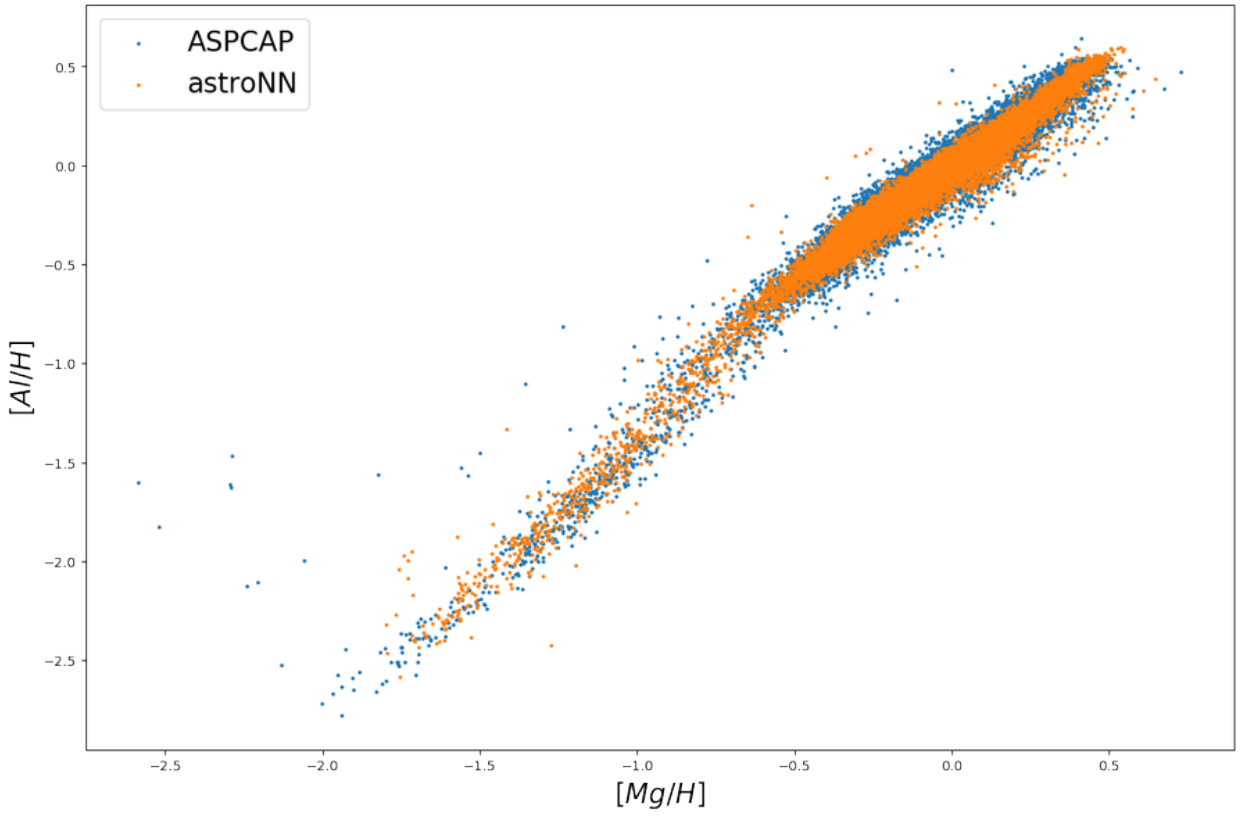


So Censored Neural Net is proposed to solve the issue by encouraging neural network to look at the ASPCAP window regions.

And it seems like it solved the issue and now neural network show a spread in $[Al/H]$ but not $[Mg/H]$



with this censored neural network and plot the training set, indeed it shows a little more spread



6.7.3 ASPCAP Labels Prediction

Internal model identifier for the author: `astroNN_0529_run010`

Training set and Testing set is exactly the same as *APOGEE Spectra with Bayesian Neural Net - astroNN.models.ApogeeBCNN*

Training set (30067 spectra + separate 3340 validation spectra): Starflag=0 and ASPCAPflag=0, 4000<Teff<5500, 200<SNR

Testing set (97723 spectra): Individual Visit of the training spectra, median SNR is around SNR~100

Using *astroNN.models.ApogeeBCNNCensored* with default hyperparameter

Ground Truth is ASPCAP labels.

	Median of residue	astropy mad_std of residue
Al	-0.002	0.047
C	0.000	0.033
C1	0.000	0.044
Ca	0.001	0.024
Co	-0.002	0.072
Cr	-0.006	0.033
Fe	-0.003	0.019
K	-0.001	0.036
Log(g)	0.006	0.049
Mg	-0.002	0.021
Mn	-0.004	0.032
N	-0.004	0.035
Na	-0.014	0.118
Ni	-0.003	0.023
O	0.001	0.033
P	0.001	0.100
S	0.000	0.048
Si	-0.002	0.024
Teff	2.310	23.296
Ti	-0.001	0.035
Ti2	-0.006	0.090
V	-0.002	0.067

6.8 APOGEE Spectra with Bayesian NN and Gaia offset calibration - astroNN.models.ApogeeDR14GaiaDR2BCNN

class astroNN.models.apogee_models.**ApogeeDR14GaiaDR2BCNN**(*lr=0.001*, *dropout_rate=0.3*)

Class for Bayesian convolutional neural network for APOGEE DR14 Gaia DR2

History 2018-Nov-06 - Written - Henry Leung (University of Toronto)

ApogeeDR14GaiaDR2BCNN can only be used with Apogee spectra with 7,514 pixels

```
from astroNN.models import ApogeeDR14GaiaDR2BCNN
from astroNN.datasets import H5Loader

# Load the train data from dataset first, x_train is spectra and y_train will be␣
↪ASPCAP labels
loader = H5Loader('datasets.h5')
loader.load_combined = True
loader.load_err = False
loader.target = ['Ks-band fakemag']
x_train, y_train, x_err, y_err = loader.load()

# And then create an instance of Apogee Censored Bayesian Convolutional Neural␣
↪Network class
apogee_gaia_bcnn = ApogeeDR14GaiaDR2BCNN()

# Set max_epochs to 10 for a quick result. You should train more epochs normally,␣
↪especially with dropout
apogee_gaia_bcnn.max_epochs = 10
apogee_gaia_bcnn.train(x_train, y_train, x_err, y_err)
```

Here is a list of parameter you can set but you can also not set them to use default

```
ApogeeDR14GaiaDR2BCNN.batch_size = 64
ApogeeDR14GaiaDR2BCNN.initializer = 'he_normal'
ApogeeDR14GaiaDR2BCNN.activation = 'relu'
ApogeeDR14GaiaDR2BCNN.num_filters = [2, 4]
ApogeeDR14GaiaDR2BCNN.filter_len = 8
ApogeeDR14GaiaDR2BCNN.pool_length = 4
# number of neurone for [old_bcnn_1, old_bcnn_2, offset_hidden_1, offset_hidden_2]
ApogeeDR14GaiaDR2BCNN.num_hidden = [162, 64, 32, 16]
ApogeeDR14GaiaDR2BCNN.max_epochs = 50
ApogeeDR14GaiaDR2BCNN.lr = 0.005
ApogeeDR14GaiaDR2BCNN.reduce_lr_epsilon = 0.00005
ApogeeDR14GaiaDR2BCNN.reduce_lr_min = 0.0000000001
ApogeeDR14GaiaDR2BCNN.reduce_lr_patience = 10
ApogeeDR14GaiaDR2BCNN.target = 'all'
ApogeeDR14GaiaDR2BCNN.l2 = 5e-9
ApogeeDR14GaiaDR2BCNN.dropout_rate = 0.2
ApogeeDR14GaiaDR2BCNN.input_norm_mode = 3
ApogeeDR14GaiaDR2BCNN.labels_norm_mode = 2
```

Note: You can disable astroNN data normalization via `ApogeeDR14GaiaDR2BCNN.input_norm_mode=0` as well as `ApogeeDR14GaiaDR2BCNN.labels_norm_mode=0` and do normalization yourself. But make sure you don't normalize labels with MAGIC_NUMBER (missing labels).

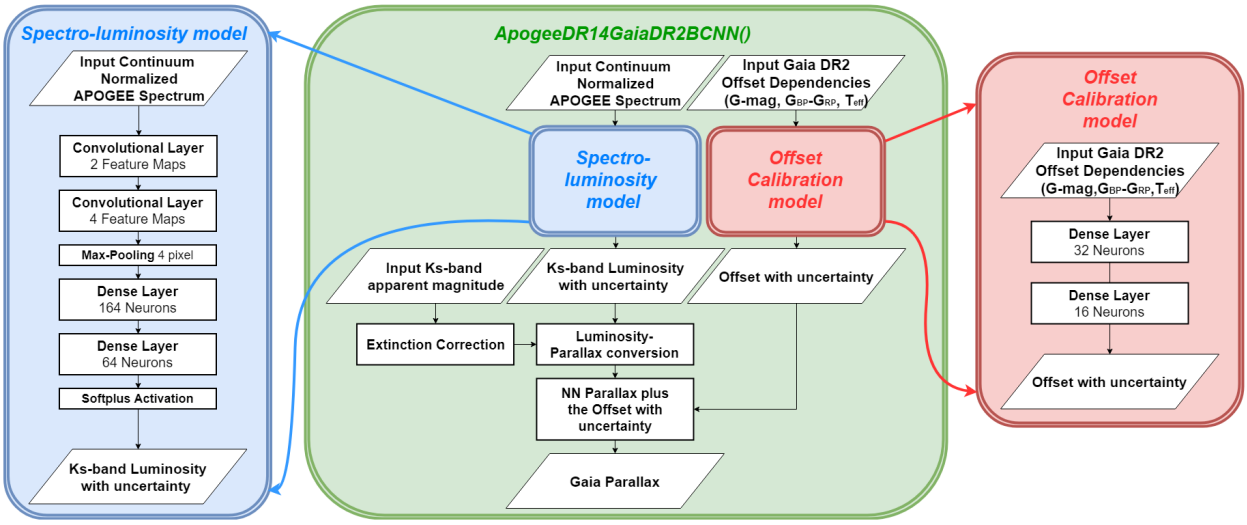
After the training, you can use *apogee_gaia_bcnn* in this case and call test method to test the neural network on test data. Or you can load the folder by

```
from astroNN.models import load_folder
apogee_gaia_bcnn = load_folder('astroNN_0101_run001')

# Load the test data from dataset, x_test is spectra and y_test will be ASPCAP labels
test_data = ......

# pred contains denormalized result aka. fakemag prediction in this case
# pred_std is a list of uncertainty
# pred_std['total'] is the total uncertainty (standard derivation) which is the sum
↪of all the uncertainty
# pred_std['predictive'] is the predictive uncertainty predicted by bayesian neural
↪net
# pred_std['model'] is the model uncertainty from dropout variational inference
pred, pred_std = apogee_gaia_bcnn.test(test_data)
```

```
# Calculate jacobian
jacobian_array = apogee_gaia_bcnn.jacobian(x_test, mean_output=True)
```

6.8.1 Architecture

The architecture of this neural network is as follow.

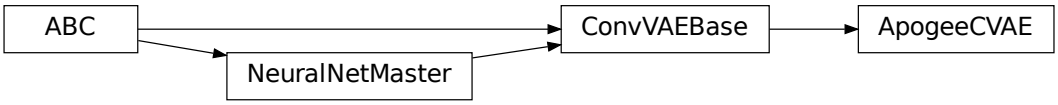


6.9 Convolutional Variational Autoencoder - astroNN.models.ApogeeCVAE

Warning: Information are obsolete, the following code may not be able to run properly with astroNN latest commit

class astroNN.models.apogee_models.**ApogeeCVAE**
Class for Convolutional Autoencoder Neural Network for stellar spectra analysis

History 2017-Dec-21 - Written - Henry Leung (University of Toronto)



It is a 9 layered convolutional neural net (2 convolutional layers->2 dense layers->latent space->2 dense layers->2 convolutional layers)

You can create ApogeeVAE via

```
from astroNN.models import ApogeeCVAE

# And then create an object of ApogeeCVAE classs
cvae_net = ApogeeCVAE()
```

6.9.1 APOGEE Spectra Analysis

Although in theory you can feed any 1D data to astroNN neural networks. This tutorial will only focus on spectra analysis.

```
from astroNN.models import ApogeeCVAE
from astroNN.datasets import H5Loader

# Load the train data from dataset first, x_train is spectra and y_train will be␣
↪ASPCAP labels
loader = H5Loader('datasets.h5')
x_train, y_train = loader.load()

# And then create an object of Bayesian Convolutional Neural Network classs
cvae_net = ApogeeCVAE()

# Set max_epochs to 10 for a quick result. You should train more epochs normally,␣
↪especially with dropout
cvae_net.max_epochs = 10
cvae_net.train(x_train)
```

After the training, you can use 'vae_net' in this case and call test method to test the neural network on test data. Or you can load the folder by

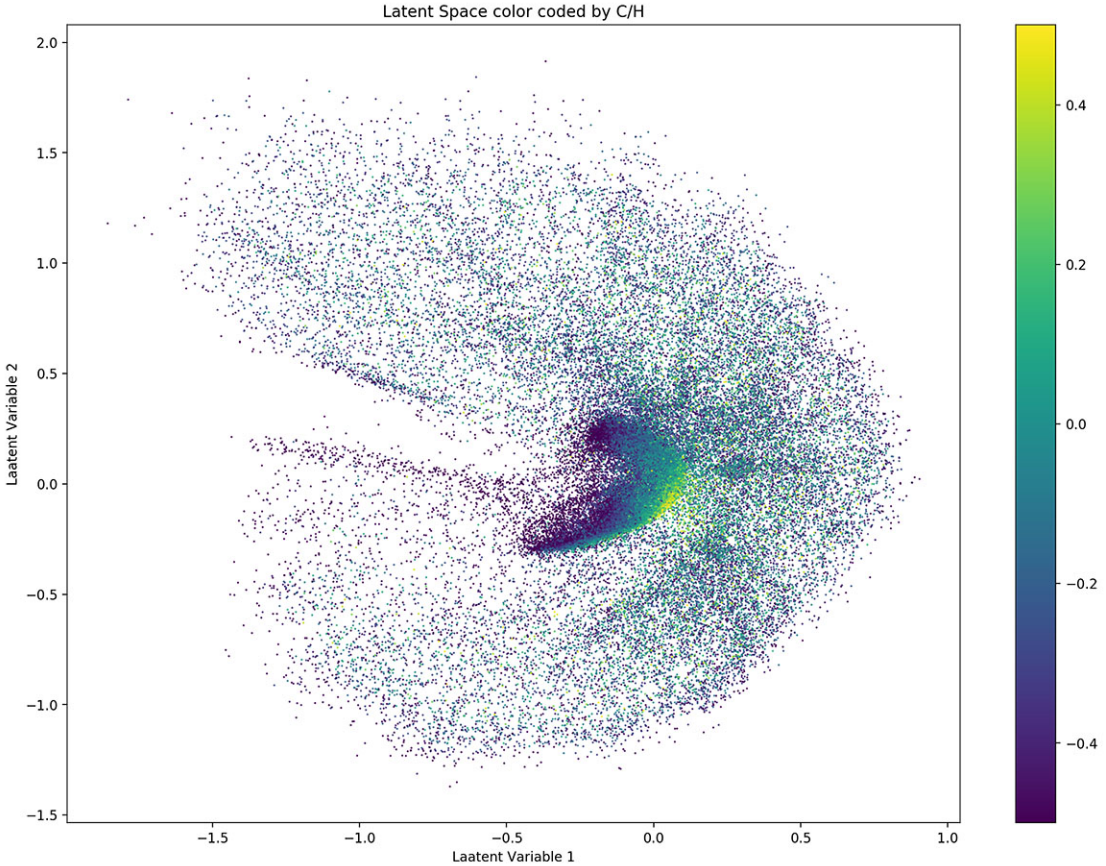
```
from astroNN.models import load_folder
cvae_net = load_folder('astroNN_0101_run001')

# Load the test data from dataset, x_test is spectra and y_test will be ASPCAP labels
loader2 = H5Loader('datasets.h5')
loader2.load_combined = False
x_test, y_test = loader2.load()
```

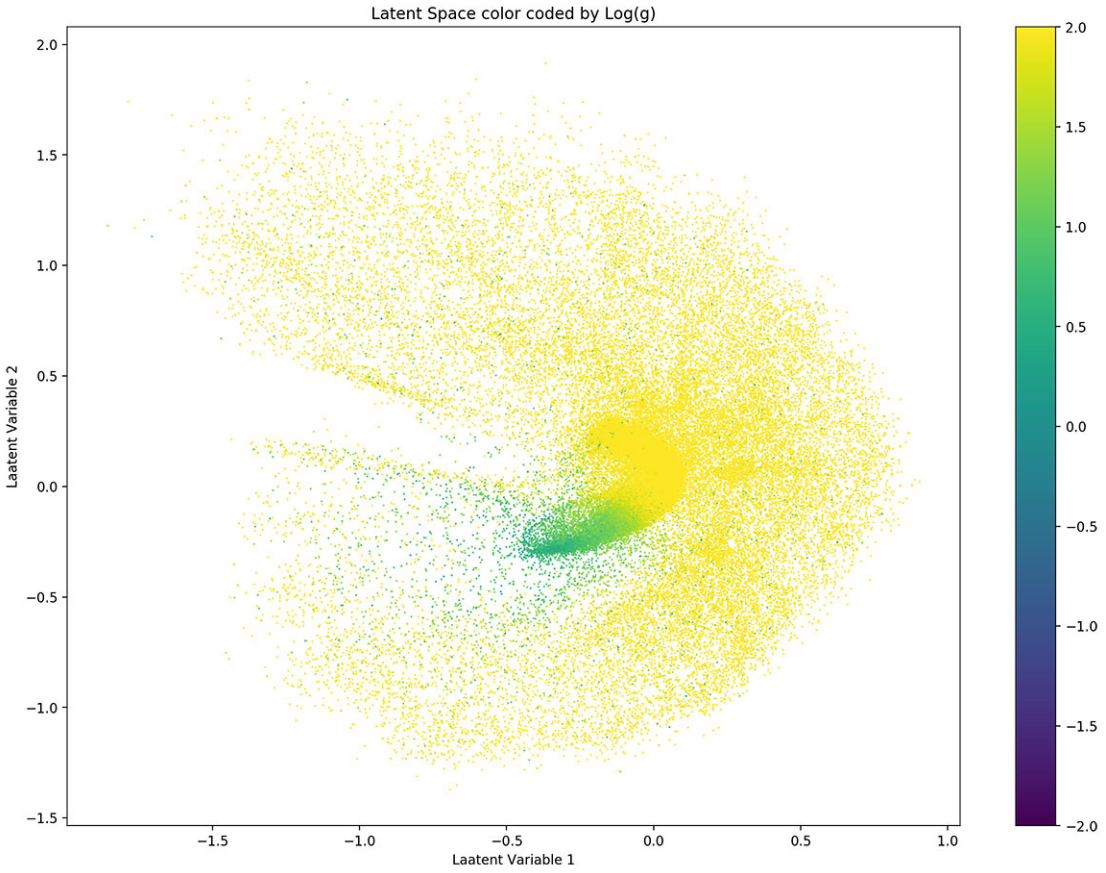
VAE is a special case. You can either use test_encoder(x_test) to get the value in latent space or use test(x_test) to get spectra reconstruction

```
# Get latent space representation
latent_space_value = cvae_net.test_encoder(x_test)

# Get spectra reconstruction
spectra_recon = cvae_net.test(x_test)
```

Note: You can access to Keras model method like model.predict via (in the above tutorial) vae_net.keras_model (Example: vae_net.keras_model.predict())

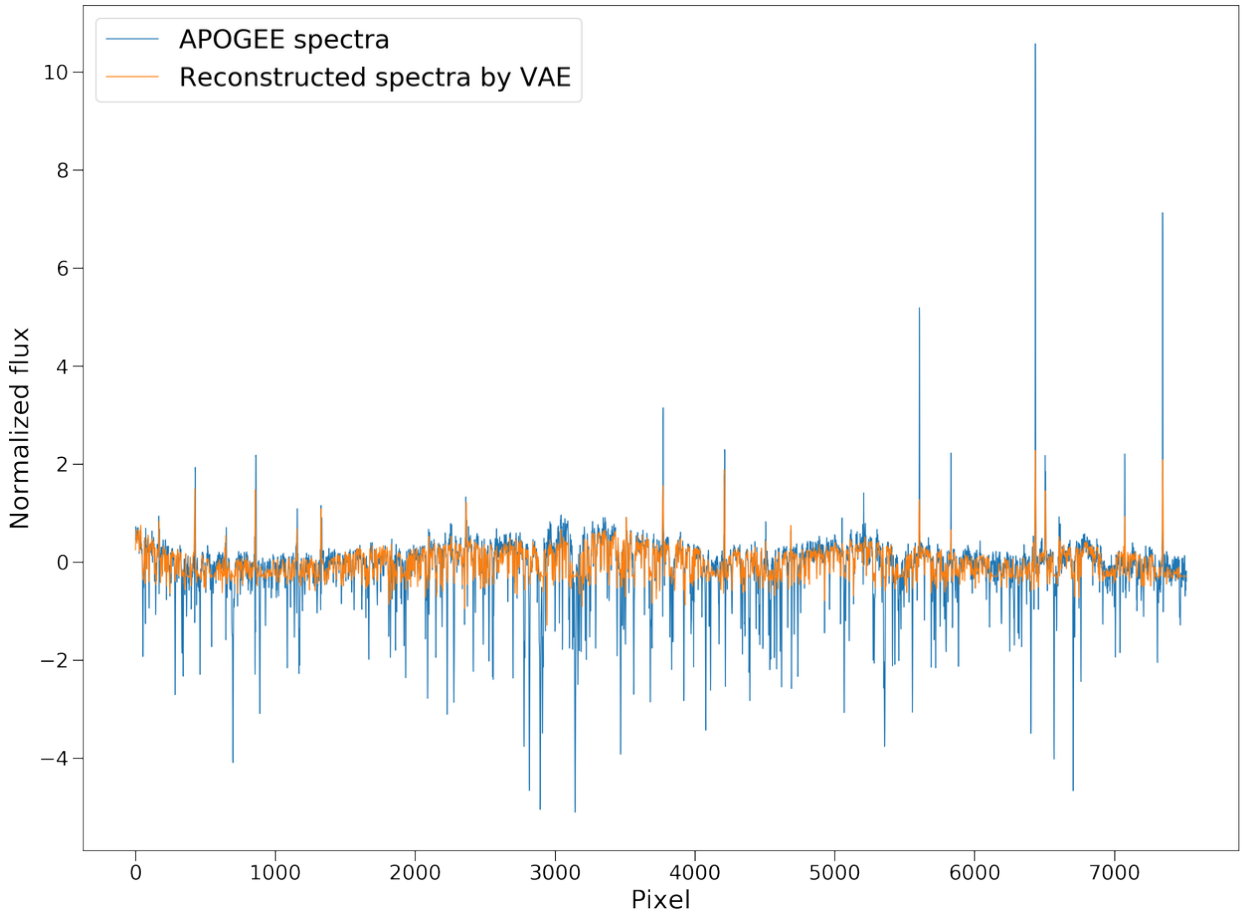
6.9.2 Example Plots on latent space using VAE.plot_latent()





6.9.3 Example Plots on spectra reconstruction

```
x_re = cvae_net.test(x_test)

import pylab as plt

fig = plt.figure(figsize=(20, 15), dpi=150)
plt.plot(x[0], linewidth=0.9, label='APOGEE spectra')
plt.plot(x_re[0], linewidth=0.9, label='Reconstructed spectra by VAE')
plt.xlabel('Pixel', fontsize=25)
plt.ylabel('Normalized flux', fontsize=25)
plt.legend(loc='best', fontsize=25)
plt.tick_params(labelsize=20, width=1, length=10)
```

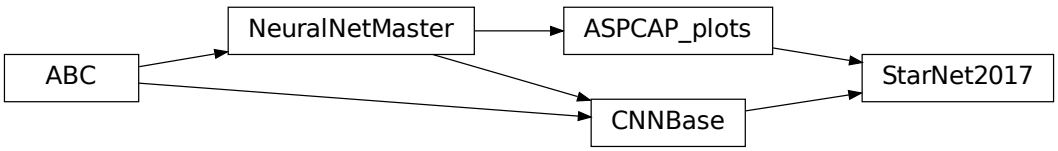


6.10 StarNet (arXiv:1709.09182)

class astroNN.models.apogee_models.**StarNet2017**

To create StarNet, S. Fabbro et al. (2017) arXiv:1709.09182. astroNN implemented the exact architecture with default parameter same as StarNet paper

History 2017-Dec-23 - Written - Henry Leung (University of Toronto)



StarNet2017 is a astroNN neural network implementation from the paper (arXiv:1709.09182), StarNet2017 is inherited from astroNN's CNNBase class defined in astroNN.models.NeuralNetBases

You can create StarNet2017 via

```
from astroNN.models import StarNet2017
from astroNN.datasets import H5Loader

# And then create an object of StarNet2017 classs
starnet_net = StarNet2017()

# Load the train data from dataset first, x_train is spectra and y_train will be␣
↪ASPCAP labels
loader = H5Loader('datasets.h5')
loader.load_err = False
x_train, y_train = loader.load()

# And then create an object of Convolutional Neural Network classs
starnet = StarNet2017()

# Set max_epochs to 10 for a quick result. You should train more epochs normally
starnet.max_epochs = 10
starnet.train(x_train, y_train)
```

Note: Default hyperparameter is the same as the original StarNet paper

6.11 Cifar10 with astroNN

Here is a Cifar10 example using astroNN

```
from keras.datasets import cifar10
from keras import utils
import numpy as np

from astroNN.models import Cifar10CNN

(x_train, y_train), (x_test, y_test) = cifar10.load_data()
y_train = utils.to_categorical(y_train, 10)
y_test = utils.to_categorical(y_test, 10)

y_train = y_train.astype(np.float32)
x_train = x_train.astype(np.float32)

x_test = x_test.astype(np.float32)
y_test = y_test.astype(np.float32)

net = Cifar10CNN()
net.max_epochs = 10
net.train(x_train, y_train)
```

```
# Load the folder back
from astroNN.models import load_folder

# Replace with correct name
cnn = load_folder('astroNN_0114_run001')
prediction = cnn.test(x_test)
print(prediction)
```

CHAPTER 7

Acknowledging astroNN

Please cite the following paper that describes astroNN if astroNN used in your research as well as consider linking it to https://github.com/henrysky/astroNN

Deep learning of multi-element abundances from high-resolution spectroscopic data [arXiv:1804.08622][ADS]

And here is a list of publications using `astroNN` - papers

CHAPTER 8

Authors

- **Henry Leung** - *Initial work and developer* - henrysky
  Astronomy Student, University of Toronto
  Contact Henry: henrysky.leung [at] mail.utoronto.ca
- **Jo Bovy** - *Project Supervisor* - jobovy
  Astronomy Professor, University of Toronto

Python Module Index

a

Index

Symbols

A

B

C

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

Z