

---

# **astroNN Documentation**

***Release 1.1.0***

**Henry Leung**

**Sep 12, 2023**



## DATASETS

<b>1</b>	<b>Indices, tables and astroNN structure</b>	<b>3</b>
<b>2</b>	<b>Acknowledging astroNN</b>	<b>151</b>
<b>3</b>	<b>Authors</b>	<b>153</b>
	<b>Python Module Index</b>	<b>155</b>
	<b>Index</b>	<b>157</b>





astroNN is a python package to do various kinds of neural networks with targeted application in astronomy by using Keras API 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. 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 DECals Dataset](#)*.



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

## 1.1 Galaxy10 DECals Dataset

Welcome! Galaxy10 DECals is a much improved version of our original [Galaxy10](#). The source code is here: <https://github.com/henrysky/Galaxy10>

The original Galaxy10 dataset was created with Galaxy Zoo (GZ) Data Release 2 where volunteers classify ~270k of SDSS galaxy images where ~22k of those images were selected in 10 broad classes using volunteer votes. GZ later utilized images from [DESI Legacy Imaging Surveys](#) (DECals) with much better resolution and image quality. Galaxy10 DECals has combined all three (GZ DR2 with DECals images instead of SDSS images and DECals campaign ab, c) results in ~441k of unique galaxies covered by DECals where ~18k of those images were selected in 10 broad classes using volunteer votes with more rigorous filtering. Galaxy10 DECals had its 10 broad classes tweaked a bit so that each class is more distinct from each other and Edge-on Disk with Boxy Bulge class with only 17 images in original Galaxy10 was abandoned. The source code for this dataset is released under this repository so you are welcome to play around if you like, otherwise you can use the compiled Galaxy10 DECals with download link below.

### 1.1.1 Download Galaxy10 DECals

Galaxy10\_DECals.h5: [https://www.astro.utoronto.ca/~hleung/shared/Galaxy10/Galaxy10\\_DECals.h5](https://www.astro.utoronto.ca/~hleung/shared/Galaxy10/Galaxy10_DECals.h5)

SHA256: 19AEFC477C41BB7F77FF07599A6B82A038DC042F889A111B0D4D98BB755C1571

Size: 2.54 GB

### 1.1.2 Introduction

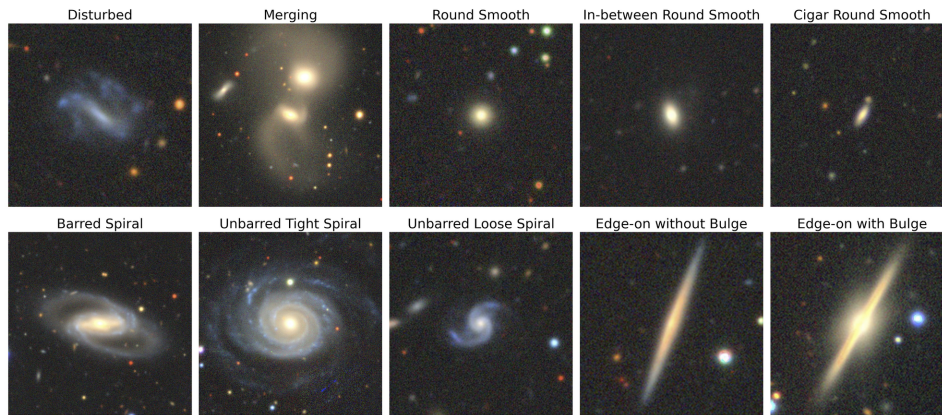
Galaxy10 DECals is a dataset contains 17736 256x256 pixels colored galaxy images (g, r and z band) separated in 10 classes. Galaxy10\_DECals.h5 have columns images with shape (17736, 256, 256, 3), ans, ra, dec, redshift and pxscale in unit of arcsecond per pixel

Galaxy10 DECals images come from [DESI Legacy Imaging Surveys](#) and labels come from [Galaxy Zoo](#).

```
Galaxy10 dataset (17736 images)
├── Class 0 (1081 images): Disturbed Galaxies
├── Class 1 (1853 images): Merging Galaxies
├── Class 2 (2645 images): Round Smooth Galaxies
├── Class 3 (2027 images): In-between Round Smooth Galaxies
├── Class 4 ( 334 images): Cigar Shaped Smooth Galaxies
├── Class 5 (2043 images): Barred Spiral Galaxies
├── Class 6 (1829 images): Unbarred Tight Spiral Galaxies
├── Class 7 (2628 images): Unbarred Loose Spiral Galaxies
├── Class 8 (1423 images): Edge-on Galaxies without Bulge
└── Class 9 (1873 images): Edge-on Galaxies with Bulge
```

For more information on the original Galaxy Zoo 2 classification tree: [Galaxy Zoo Decision Tree](#)

### Example images of each class from Galaxy10 DECals



Galaxy10 DECals: Henry Leung/Jo Bovy 2021, Data: DECals/Galaxy Zoo

### 1.1.3 Load with astroNN

```

1 from astroNN.datasets import load_galaxy10
2 from tensorflow.keras import utils
3 import numpy as np
4
5 # To load images and labels (will download automatically at the first time)
6 # First time downloading location will be ~/.astroNN/datasets/
7 images, labels = load_galaxy10()
8
9 # To convert the labels to categorical 10 classes
10 labels = utils.to_categorical(labels, 10)
11
12 # To convert to desirable type
13 labels = labels.astype(np.float32)
14 images = images.astype(np.float32)

```

### 1.1.4 OR Load with Python & h5py

You should download Galaxy10\_DECals.h5 first and open python at the same location and run the following to open it:

```

1 import h5py
2 import numpy as np
3 from tensorflow.keras import utils
4
5 # To get the images and labels from file
6 with h5py.File('Galaxy10_DECals.h5', 'r') as F:
7     images = np.array(F['images'])
8     labels = np.array(F['ans'])
9
10 # To convert the labels to categorical 10 classes
11 labels = utils.to_categorical(labels, 10)
12
13 # To convert to desirable type

```

(continues on next page)

(continued from previous page)

```
14 labels = labels.astype(np.float32)
15 images = images.astype(np.float32)
```

### 1.1.5 Split into train and test set

```
1 import numpy as np
2 from sklearn.model_selection import train_test_split
3
4 train_idx, test_idx = train_test_split(np.arange(labels.shape[0]), test_size=0.1)
5 train_images, train_labels, test_images, test_labels = images[train_idx], labels[train_
  ↪idx], images[test_idx], labels[test_idx]
```

### 1.1.6 Lookup Galaxy10 Class

You can lookup Galaxy10 class to the corresponding name by

```
1 from astroNN.datasets.galaxy10 import galaxy10cls_lookup
2 galaxy10cls_lookup(# a class number here to get back the name)
```

### 1.1.7 Galaxy10 Dataset Authors

- **Henry Leung** - [henrysky](#)  
Department of Astronomy & Astrophysics, University of Toronto
- **Jo Bovy** - [jobovy](#)  
Department of Astronomy & Astrophysics, University of Toronto

### 1.1.8 Acknowledgments

For astroNN acknowledgment, please refers to *Acknowledging astroNN*

1. Galaxy10 dataset classification labels come from [Galaxy Zoo](#)
2. Galaxy10 dataset images come from [DESI Legacy Imaging Surveys](#)

Galaxy Zoo is described in [Lintott et al. 2008](#), the GalaxyZoo Data Release 2 is described in [Lintott et al. 2011](#), Galaxy Zoo DECaLS Campaign is described in [Walmsley M. et al. 2021](#), DESI Legacy Imaging Surveys is described in [Dey A. et al., 2019](#)

The Legacy Surveys consist of three individual and complementary projects: the Dark Energy Camera Legacy Survey (DECaLS; Proposal ID #2014B-0404; PIs: David Schlegel and Arjun Dey), the Beijing-Arizona Sky Survey (BASS; NAOJ Prop. ID #2015A-0801; PIs: Zhou Xu and Xiaohui Fan), and the Mayall z-band Legacy Survey (MzLS; Prop. ID #2016A-0453; PI: Arjun Dey). DECaLS, BASS and MzLS together include data obtained, respectively, at the Blanco telescope, Cerro Tololo Inter-American Observatory, NSF’s NOIRLab; the Bok telescope, Steward Observatory, University of Arizona; and the Mayall telescope, Kitt Peak National Observatory, NOIRLab. The Legacy Surveys project is honored to be permitted to conduct astronomical research on Iolkam Du’ag (Kitt Peak), a mountain with particular significance to the Tohono O’odham Nation.

### 1.1.9 Some papers that used Galaxy 10

1. DeepAstroUDA: Semi-Supervised Universal Domain Adaptation for Cross-Survey Galaxy Morphology Classification and Anomaly Detection  
Aleksandra Ćiprijanović et al (2023)  
[arXiv:2302.02005](#)
2. Semi-Supervised Domain Adaptation for Cross-Survey Galaxy Morphology Classification and Anomaly Detection  
Aleksandra Ćiprijanović et al (2022)  
[arXiv:2211.00677](#)
3. Equivariance-aware Architectural Optimization of Neural Networks  
Kaitlin Maile et al (2022)  
[arXiv:2210.05484](#)
4. Galaxy Morphology Classification with DenseNet  
Wuyu Hui et al (2022)  
[JPhCS, 2402, 012009](#)
5. Learning to Detect Interesting Anomalies  
Alireza Vafaei Sadr et al (2022)  
[arXiv:2210.16334](#)
6. Machine learning in introductory astrophysics laboratory activities  
Alireza Vafaei Sadr (2022)  
[PhTea, 59, 662](#)
7. Physics-Integrated Variational Autoencoders for Robust and Interpretable Generative Modeling  
Naoya Takeishi et al (2021)  
[arXiv:2102.13156](#)
8. SetGAN: Improving the stability and diversity of generative models through a permutation invariant architecture  
Alessandro Ferrero et al (2019)  
[arXiv:1907.00109](#)
9. Input Selection for Bandwidth-Limited Neural Network Inference  
Stefan Oehmcke et al (2019)  
[arXiv:1906.04673](#)

## 1.2 Galaxy10 SDSS Dataset

---

**Note:** This page has been renamed to Galaxy10 SDSS. A new Galaxy10 using DECals is available at <https://github.com/henrysky/Galaxy10>

---

### 1.2.1 Introduction

Galaxy10 SDSS is a dataset contains 21785 69x69 pixels colored galaxy images (g, r and i band) separated in 10 classes. Galaxy10 SDSS images come from [Sloan Digital Sky Survey](#) and labels come from [Galaxy Zoo](#).

```
Galaxy10 dataset (21785 images)
— Class 0 (3461 images): Disk, Face-on, No Spiral
— Class 1 (6997 images): Smooth, Completely round
— Class 2 (6292 images): Smooth, in-between round
— Class 3 (394 images): Smooth, Cigar shaped
— Class 4 (1534 images): Disk, Edge-on, Rounded Bulge
— Class 5 (17 images): Disk, Edge-on, Boxy Bulge
— Class 6 (589 images): Disk, Edge-on, No Bulge
— Class 7 (1121 images): Disk, Face-on, Tight Spiral
— Class 8 (906 images): Disk, Face-on, Medium Spiral
— Class 9 (519 images): Disk, Face-on, Loose Spiral
```

These classes are mutually exclusive, but Galaxy Zoo relies on human volunteers to classify galaxy images and the volunteers do not agree on all images. For this reason, Galaxy10 only contains images for which more than 55% of the votes agree on the class. That is, more than 55% of the votes among 10 classes are for a single class for that particular image. If none of the classes get more than 55%, the image will not be included in Galaxy10 as no agreement was reached. As a result, 21785 images after the cut.

The justification of 55% as the threshold is based on validation. Galaxy10 is meant to be an alternative to MNIST or Cifar10 as a deep learning toy dataset for astronomers. Thus `astroNN.models.Cifar10_CNN` is used with Cifar10 as a reference. The validation was done on the same `astroNN.models.Cifar10_CNN`. 50% threshold will result a poor neural network classification accuracy although around 36000 images in the dataset, many are probably misclassified and neural network has a difficult time to learn. 60% threshold result is similar to 55% , both classification accuracy is similar to Cifar10 dataset on the same network, but 55% threshold will have more images be included in the dataset. Thus 55% was chosen as the threshold to cut data.

The original images are 424x424, but were cropped to 207x207 centered at the images and then downsampled 3 times via bilinear interpolation to 69x69 in order to make them manageable on most computer and graphics card memory.

There is no guarantee on the accuracy of the labels. Moreover, Galaxy10 is not a balanced dataset and it should only be used for educational or experimental purpose. If you use Galaxy10 for research purpose, please cite Galaxy Zoo and Sloan Digital Sky Survey.

For more information on the original classification tree: [Galaxy Zoo Decision Tree](#)



## Example images of each class from Galaxy10 dataset



Galaxy10 Dataset: Henry Leung/Jo Bovy 2018, Data Source: SDSS/Galaxy Zoo

### 1.2.2 Download Galaxy10 SDSS

Galaxy10.h5: <http://www.astro.utoronto.ca/~bovy/Galaxy10/Galaxy10.h5>

SHA256: 969A6B1CEFC36E09FFFA86FEBD2F699A4AA19B837BA0427F01B0BC6DED458AF

Size: 200 MB (210,234,548 bytes)

Or see below to load (and download automatically) the dataset with astroNN

### 1.2.3 TL;DR for Beginners

You can view the Jupyter notebook in here: [https://github.com/henrysky/astroNN/blob/master/demo\\_tutorial/galaxy10/Galaxy10\\_Tutorial.ipynb](https://github.com/henrysky/astroNN/blob/master/demo_tutorial/galaxy10/Galaxy10_Tutorial.ipynb)

OR you can train with astroNN and just copy and paste the following script to get and train a simple neural network on Galaxy10

Basically first we load the Galaxy10 with astroNN and split into train and test set. astroNN will split the training set into training data and validation data as well as normalizing them automatically.

*Galaxy10CNN* is a simple 4 layered convolutional neural network consisted of 2 convolutional layers and 2 dense layers.

```

1 # import everything we need first
2 from tensorflow.keras import utils
3 import numpy as np
4 from sklearn.model_selection import train_test_split
5 import pylab as plt
6
7 from astroNN.models import Galaxy10CNN
8 from astroNN.datasets import load_galaxy10sdss
9 from astroNN.datasets.galaxy10sdss import galaxy10cls_lookup, galaxy10_confusion
10
11 # To load images and labels (will download automatically at the first time)
12 # First time downloading location will be ~/.astroNN/datasets/
13 images, labels = load_galaxy10sdss()
```

(continues on next page)

(continued from previous page)

```

14
15 # To convert the labels to categorical 10 classes
16 labels = utils.to_categorical(labels, 10)
17
18 # Select 10 of the images to inspect
19 img = None
20 plt.ion()
21 print('=====Data Inspection=====')
22 for counter, i in enumerate(range(np.random.randint(0, labels.shape[0], size=10).
↳shape[0])):
23     img = plt.imshow(images[i])
24     plt.title('Class {}: {} \n Random Demo images {} of 10'.format(np.argmax(labels[i]),
↳galaxy10cls_lookup(labels[i]), counter+1))
25     plt.draw()
26     plt.pause(2.)
27 plt.close('all')
28 print('=====Data Inspection Finished=====')
29
30 # To convert to desirable type
31 labels = labels.astype(np.float32)
32 images = images.astype(np.float32)
33
34 # Split the dataset into training set and testing set
35 train_idx, test_idx = train_test_split(np.arange(labels.shape[0]), test_size=0.1)
36 train_images, train_labels, test_images, test_labels = images[train_idx], labels[train_
↳idx], images[test_idx], labels[test_idx]
37
38 # To create a neural network instance
39 galaxy10net = Galaxy10CNN()
40
41 # set maximum epochs the neural network can run, set 5 to get quick result
42 galaxy10net.max_epochs = 5
43
44 # To train the nerual net
45 # astroNN will normalize the data by default
46 galaxy10net.train(train_images, train_labels)
47
48 # print model summary before training
49 galaxy10net.keras_model.summary()
50
51 # After the training, you can test the neural net performance
52 # Please notice predicted_labels are labels predicted from neural network. test_labels_
↳are ground truth from the dataset
53 predicted_labels = galaxy10net.test(test_images)
54
55 # Convert predicted_labels to class
56 prediction_class = np.argmax(predicted_labels, axis=1)
57
58 # Convert test_labels to class
59 test_class = np.argmax(test_labels, axis=1)
60
61 # Prepare a confusion matrix

```

(continues on next page)

(continued from previous page)

```

62 confusion_matrix = np.zeros((10,10))
63
64 # create the confusion matrix
65 for counter, i in enumerate(prediction_class):
66     confusion_matrix[i, test_class[counter]] += 1
67
68 # Plot the confusion matrix
69 galaxy10_confusion(confusion_matrix)

```

### 1.2.4 Load with astroNN

```

1  from astroNN.datasets import load_galaxy10sdss
2  from tensorflow.keras import utils
3  import numpy as np
4
5  # To load images and labels (will download automatically at the first time)
6  # First time downloading location will be ~/.astroNN/datasets/
7  images, labels = load_galaxy10sdss()
8
9  # To convert the labels to categorical 10 classes
10 labels = utils.to_categorical(labels, 10)
11
12 # To convert to desirable type
13 labels = labels.astype(np.float32)
14 images = images.astype(np.float32)

```

### 1.2.5 OR Load with Python & h5py

You should download Galaxy10.h5 first and open python at the same location and run the following to open it:

```

1  import h5py
2  import numpy as np
3  from tensorflow.keras import utils
4
5  # To get the images and labels from file
6  with h5py.File('Galaxy10.h5', 'r') as F:
7      images = np.array(F['images'])
8      labels = np.array(F['ans'])
9
10 # To convert the labels to categorical 10 classes
11 labels = utils.to_categorical(labels, 10)
12
13 # To convert to desirable type
14 labels = labels.astype(np.float32)
15 images = images.astype(np.float32)

```

## 1.2.6 Split into train and test set

```
1 import numpy as np
2 from sklearn.model_selection import train_test_split
3
4 train_idx, test_idx = train_test_split(np.arange(labels.shape[0]), test_size=0.1)
5 train_images, train_labels, test_images, test_labels = images[train_idx], labels[train_idx], images[test_idx], labels[test_idx]
```

## 1.2.7 Lookup Galaxy10 Class

You can lookup Galaxy10 class to the corresponding name by

```
1 from astroNN.datasets.galaxy10sdss import galaxy10cls_lookup
2 galaxy10cls_lookup(# a class number here to get back the name)
```

## 1.2.8 Galaxy10 Dataset Authors

- **Henry Leung** - Compile the Galaxy10 - [henrysky](#)  
Department of Astronomy & Astrophysics, University of Toronto
- **Jo Bovy** - Supervisor of Henry Leung - [jobovy](#)  
Department of Astronomy & Astrophysics, University of Toronto

## 1.2.9 Acknowledgments

For astroNN acknowledgment, please refers to *Acknowledging astroNN*

1. Galaxy10 dataset classification labels come from [Galaxy Zoo](#)
2. Galaxy10 dataset images come from [Sloan Digital Sky Survey \(SDSS\)](#)

Galaxy Zoo is described in [Lintott et al. 2008, MNRAS, 389, 1179](#) and the data release is described in [Lintott et al. 2011, 410, 166](#)

Funding for the SDSS and SDSS-II has been provided by the Alfred P. Sloan Foundation, the Participating Institutions, the National Science Foundation, the U.S. Department of Energy, the National Aeronautics and Space Administration, the Japanese Monbukagakusho, the Max Planck Society, and the Higher Education Funding Council for England. The SDSS Web Site is <http://www.sdss.org/>.

The SDSS is managed by the Astrophysical Research Consortium for the Participating Institutions. The Participating Institutions are the American Museum of Natural History, Astrophysical Institute Potsdam, University of Basel, University of Cambridge, Case Western Reserve University, University of Chicago, Drexel University, Fermilab, the Institute for Advanced Study, the Japan Participation Group, Johns Hopkins University, the Joint Institute for Nuclear Astrophysics, the Kavli Institute for Particle Astrophysics and Cosmology, the Korean Scientist Group, the Chinese Academy of Sciences (LAMOST), Los Alamos National Laboratory, the Max-Planck-Institute for Astronomy (MPIA), the Max-Planck-Institute for Astrophysics (MPA), New Mexico State University, Ohio State University, University of Pittsburgh, University of Portsmouth, Princeton University, the United States Naval Observatory, and the University of Washington.

## 1.3 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 to install:

```
$ python -m pip install .
```

or to develop:

```
$ python -m pip install -e .
```

### 1.3.1 Prerequisites

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

Python 3.7 or above

Tensorflow (the latest version is recommended)

Tensorflow-Probability (the latest version is recommended)

CUDA and CuDNN (optional)

graphviz and pydot are required to plot the model architecture

scikit-learn, tqdm, pandas, h5py and astroquery required for astroNN functions

Since [Tensorflow](#) and [Tensorflow-Probability](#) are rapidly developing packages and astroNN heavily depends on Tensorflow. The support policy of astroNN to these packages is only the last two official versions are supported (i.e. the latest version and the second latest version are included in test suite). Generally using the latest version of Tensorflow, Tensorflow-Probability are recommended. The current supporting status (i.e. included in test suites) are

Tensorflow 2.12.x (correspond to Tensorflow-Probability 0.19.x)

Tensorflow 2.11.x (correspond to Tensorflow-Probability 0.19.x)

**Note:** Due to bugs in Tensorflow 1.12.x: <https://github.com/tensorflow/tensorflow/issues/22952>, 1.14.x: <https://github.com/tensorflow/tensorflow/issues/27543> or 2.5.x: <https://github.com/tensorflow/tensorflow/pull/47957>, you have to patch a few lines in order for astroNN to work properly. You can patch Tensorflow by running the following code

```
from astroNN.config import tf_patch
```

```
tf_patch()
```

You can also unpatch Tensorflow to undo changes made by astroNN by running the following code

```
from astroNN.config import tf_unpatch
```

```
tf_unpatch()
```

For instruction on how to install Tensorflow, please refers to their official website [Installing TensorFlow](#)

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](https://en.wikipedia.org/wiki/Advanced_Vector_Extensions#CPUs_with_AVX2))  
16GB RAM **or** above  
NVIDIA Graphics card (Optional, GTX 10 series **or** above) **or** Apple Silicon  
(If using NVIDIA GPU): At least 4GB VRAM on GPU

### 1.3.2 Using astroNN on Google Colab

To use the latest commit of astroNN on Google colab, you can copy and paste the following

```
$ !pip install tensorflow
$ !pip install tensorflow_probability
$ !pip install git+https://github.com/henrysky/astroNN.git
```

### 1.3.3 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 \[at\] utoronto.ca](mailto:henrysky.leung@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](https://graphviz.gitlab.io/_pages/Download/Download_windows.html) to download the Windows package and add the package to the PATH environment variable.

### 1.3.4 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

[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.

`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* meaning no additional path will be searched when loading model. Or for example: `/users/astroNN/custom_models/;/local/some_other_custom_models/` if you have self defined model in those locations.

`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:

```
1 from astroNN.config import config_path
2
3 # astroNN will reset the config file if the flag = 2
4 config_path(flag=2)
```

### 1.3.5 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
│   │   │           ├── 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/
│   ├── LAM05_2MS_AP9_SD14_UC4_PS1_AW_Carlin_M.fits
│   └── 20111024
```

(continues on next page)

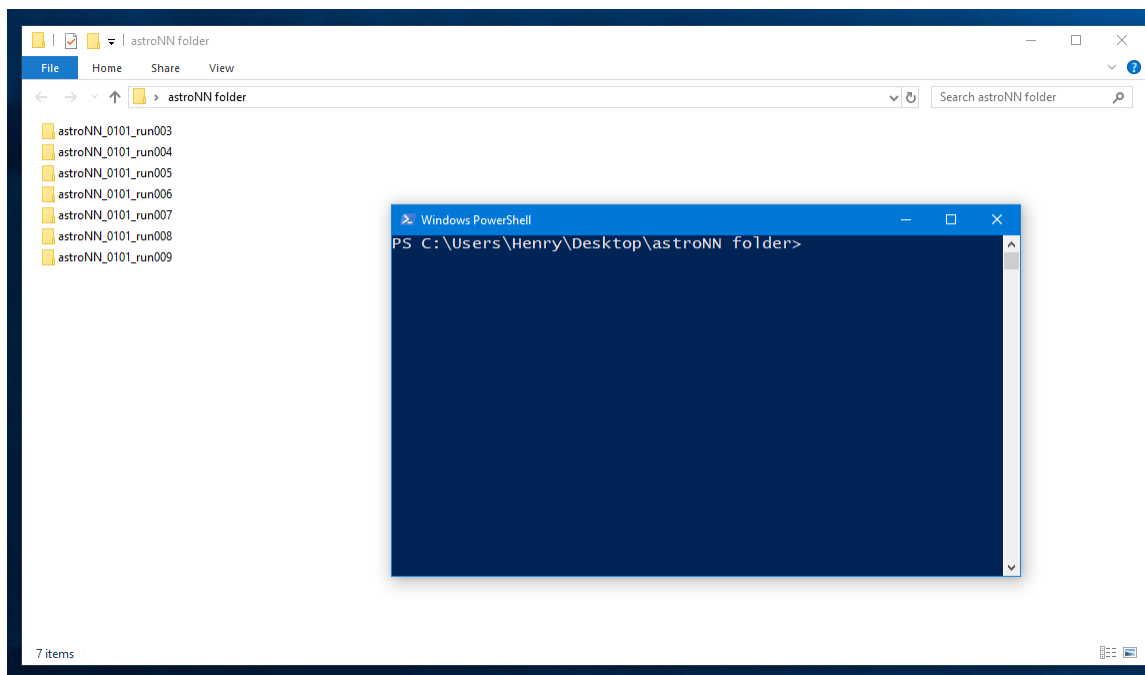


(continued from previous page)

```
├── 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



## 1.4 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.

### 1.4.1 Submitting bug reports and feature requests

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

### 1.4.2 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.

### 1.4.3 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\source\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.

### 1.4.4 Possible New Features and Improvement in the future

#### GPU/performance related issues

- Data reduction pipeline on GPU?
- Multiple GPU support!
- Training on large dataset that cant fit into memory?

**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

## 1.5 History

### 1.5.1 v1.1 series

**v1.1.0 (26 April 2023)**

This release mainly targeted to the paper A variational encoder-decoder approach to precise spectroscopic age estimation for large Galactic surveys available at [\[arXiv:2302.05479\]](https://arxiv.org/abs/2302.05479) [\[ADS\]](#)

**New features:**

- Added models: ApogeeKplerEchelle and ApokascEncoderDecoder
- Input data can now be a dict, such as `nn.train({'input': input_data, 'input': aux_input_data}, {'output': labels, 'output_aux': aux_labels})`
- Added numerical integrator for NeuralODE
- tqdm progress bar for model prediction
- Added a new improved version Galaxy10
- Added multiple metrics based on median
- Added functions `transfer_weights` for transfer learning

**Improvement:**

- Fully compatible with Tensorflow 2
- Model training/inference should be much faster by using Tensorflow v2 eager execution (see: <https://github.com/tensorflow/tensorflow/issues/33024#issuecomment-551184305>)
- Improved continuous integration testing with Github Actions, now actually test models learn properly with real world data instead of checking no syntax error with random data
- Support `sample_weight` in all loss functions and training
- Improved catalog coordinates matching
- New documentation webpages
- ~15% faster in Bayesian neural network inference by using parallelized loop
- Loss/metrics functions and normalizer now check for NaN too
- Updated many of notebooks to be compatible with the latest Tensorflow

#### Breaking Changes:

- Deprecated support for all Tensorflow 1.x
- Tested with Tensorflow 2.11 and 2.12
- Python 3.8 or above only
- Incompatible to Tensorflow 1.x and <=2.3 due to necessary changes for Tensorflow eager execution API
- Renamed neural network models `train()`, `test()`, `train_on_batch()` method to `fit()`, `predict()`, `fit_on_batch()`
- Old Galaxy10 has been renamed to Galaxy10 SDSS and the new version will replace and call Galaxy10

## 1.5.2 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

### 1.5.3 v0.0 series

**v0.0.0 (13 October 2017)**

First commit of astroNN on Github!!!

## 1.6 Publications using astroNN

1. Deep learning of multi-element abundances from high-resolution spectroscopic data  
Henry W. Leung, Jo Bovy (2019)  
[MNRAS, 483, 3255L](#)  
Original astroNN paper
2. Dynamical heating across the Milky Way disc using APOGEE and Gaia  
J. Ted Mackereth, Jo Bovy, Henry W. Leung, et al. (2019)  
[MNRAS, 489, 176M](#)  
Use ApogeeBCNN to infer spectroscopic age
3. Simultaneous calibration of spectro-photometric distances and the Gaia DR2 parallax zero-point offset with deep learning  
Henry W. Leung, Jo Bovy (2019)  
[MNRAS, 489, 2079L](#)  
Use ApogeeDR14GaiaDR2BCNN to infer spectro-photometric distances
4. Solar image denoising with convolutional neural networks  
C. J. Díaz Baso, J. de la Cruz Rodríguez, S. Danilovic (2019)  
[A&A, 629, 99D](#)
5. A variational encoder-decoder approach to precise spectroscopic age estimation for large Galactic surveys  
Henry W. Leung, Jo Bovy, J. Ted Mackereth, Andrea Miglio (2023)  
[arXiv:2302.05479](#)  
Use ApokascEncoderDecoder to infer spectroscopic age trained on APOGEE and Kepler

### 1.6.1 Publication figure style

astroNN contains a function that helps me to standardize matplotlib figure style used in my publication.

```
astroNN.shared.matplotlib.pylab_style(paper=False)
```

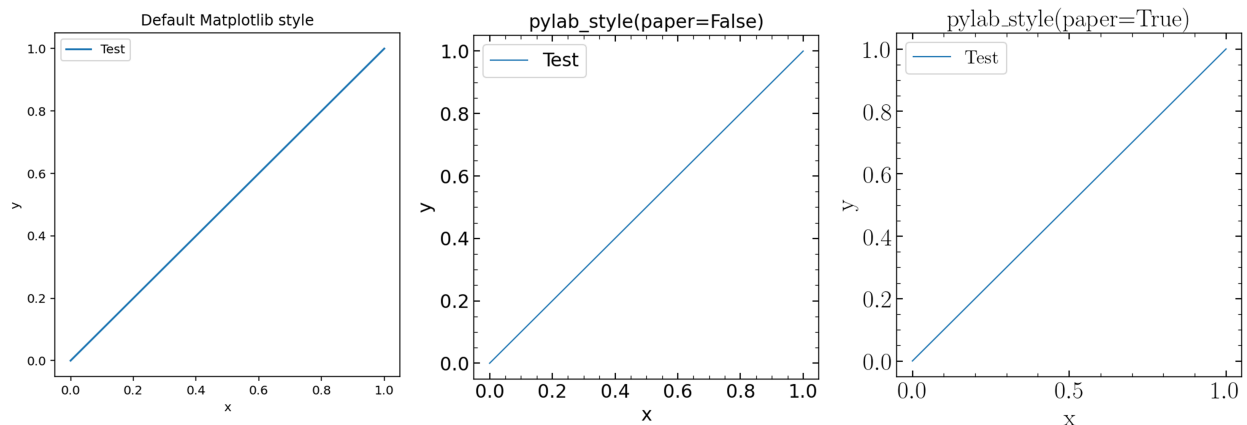
Such function can be used by simply calling it before using matplotlib to plot any figure

```
1 import matplotlib.pyplot as plt
2 from astroNN.shared import pylab_style
3
4 pylab_style(paper=True)
5
6 # matplotlib code goes here
```

If you do not have  $\text{\LaTeX}$  installed on your computer, you can set the paper option to False like `pylab_style(paper=False)`

Here are a figure that compare different styles using the following matplotlib code

```
1 plt.figure(figsize=(5, 5))
2 plt.plot([0, 1], [0, 1], label="Test")
3 plt.xlabel("x")
4 plt.ylabel("y")
5 plt.legend()
```



## 1.7 Loss Functions and Metrics

astroNN provides modified loss functions under `astroNN.nn.losses` module 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

### 1.7.1 Correction Term for Magic Number

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

Calculate a correction term to prevent the loss being “lowered” by magic\_num or NaN

**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

### 1.7.2 Mean Squared Error

`astroNN.nn.losses.mean_squared_error(y_true, y_pred, sample_weight=None)`

Calculate mean square error losses

**Parameters**

- **y\_true** (*Union(tf.Tensor, tf.Variable)*) – Ground Truth
- **y\_pred** (*Union(tf.Tensor, tf.Variable)*) – Prediction
- **sample\_weight** (*Union(tf.Tensor, tf.Variable, list)*) – Sample weights

**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

```
1 def keras_model():
2     # Your keras_model define here
3     return model
4
5 model = keras_model()
6 # remember to import astroNN's loss function first
7 model.compile(loss=mean_squared_error, ...)
```

### 1.7.3 Mean Absolute Error

`astroNN.nn.losses.mean_absolute_error(y_true, y_pred, sample_weight=None)`

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
- **sample\_weight** (*Union(tf.Tensor, tf.Variable, list)*) – Sample weights

#### 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

```
1 def keras_model():
2     # Your keras_model define here
3     return model
4
5 model = keras_model()
6 # remember to import astroNN's loss function first
7 model.compile(loss=mean_absolute_error, ...)
```



### 1.7.4 Mean Error

`astroNN.nn.losses.mean_error(y_true, y_pred, sample_weight=None)`

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
- **sample\_weight** (*Union*(*tf.Tensor*, *tf.Variable*, *list*)) – Sample weights

#### 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

```

1 def keras_model():
2     # Your keras_model define here
3     return model
4
5 model = keras_model()
6 # remember to import astroNN's loss function first
7 model.compile(loss=mean_error, ...)
```

### 1.7.5 Regression Loss and Predictive Variance Loss for Bayesian Neural Net

`astroNN.nn.losses.robust_mse(y_true, y_pred, variance, labels_err, sample_weight=None)`

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
- **sample\_weight** (*Union*(*tf.Tensor*, *tf.Variable*, *list*)) – Sample weights

**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  $\log((\sigma_{\text{predictive},i})^2 + (\sigma_{\text{known},i})^2)$ . 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

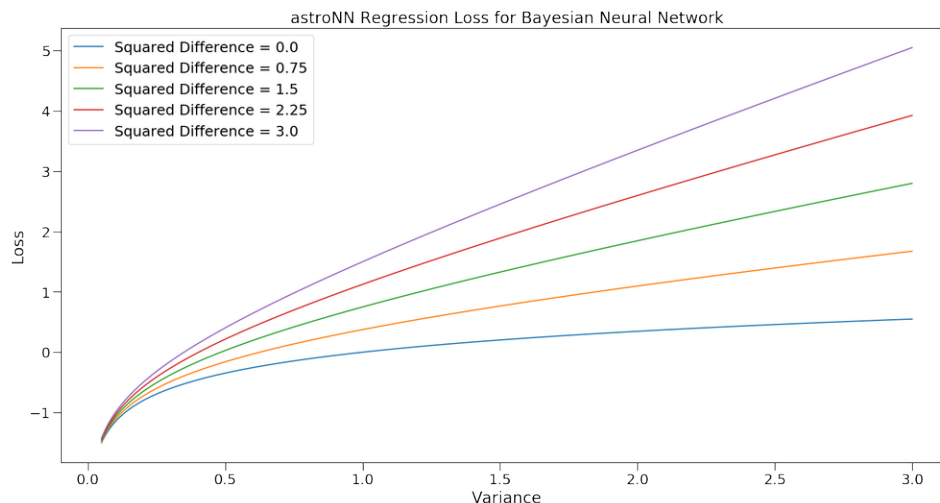
$$Loss_{BNN} = \frac{1}{D} \sum_{i=1}^{batch} (Loss_i \mathcal{F}_{\text{correction},i})$$

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

```

1 def keras_model():
2     # Your keras_model define here
3
4     # model for the training process
5     model = Model(inputs=[input_tensor, labels_err_tensor], outputs=[output, variance_
6         ↪ output])
7
8     # model for the prediction
9     model_prediction = Model(inputs=input_tensor, outputs=[output, variance_output])
10
11     variance_output = Dense(name='variance_output', ...)
12     output = Dense(name='output', ...)
13
14     predictive_variance_loss = mse_var_wrapper(output, labels_err_tensor)
15     output_loss = mse_lin_wrapper(predictive_variance, labels_err_tensor)
16
17     return model, model_prediction, output_loss, predictive_variance_loss
18
19 model, model_prediction, output_loss, predictive_variance_loss = keras_model()
20 # remember to import astroNN loss function first
21 model.compile(loss={'output': output_loss, 'variance_output': predictive_variance_loss}, ↪
22     ↪ ...)
```

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



## 1.7.6 Mean Squared Logarithmic Error

`astroNN.nn.losses.mean_squared_logarithmic_error(y_true, y_pred, sample_weight=None)`

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
- **sample\_weight** (*Union(tf.Tensor, tf.Variable, list)*) – Sample weights

### 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  $\epsilon$  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

```
1 def keras_model():
2     # Your keras_model define here
3     return model
4
5 model = keras_model()
6 # remember to import astroNN's loss function first
7 model.compile(loss=mean_squared_logarithmic_error, ...)
```

## 1.7.7 Mean Absolute Percentage Error

`astroNN.nn.losses.mean_absolute_percentage_error(y_true, y_pred, sample_weight=None)`

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

- **sample\_weight** (*Union(tf.Tensor, tf.Variable, list)*) – Sample weights

**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  $\epsilon$  is a small constant

Then Mean Absolute Percentage Error is based on the equation

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

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

```

1 def keras_model():
2     # Your keras_model define here
3     return model
4
5 model = keras_model()
6 # remember to import astroNN's loss function first
7 model.compile(loss=mean_absolute_percentage_error, ...)
```

## 1.7.8 Mean Percentage Error

`astroNN.nn.losses.mean_percentage_error(y_true, y_pred, sample_weight=None)`

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
- **sample\_weight** (*Union(tf.Tensor, tf.Variable, list)*) – Sample weights

**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  $\epsilon$  is a small constant

Then Mean Percentage Error is based on the equation

$$Loss_i = \begin{cases} 100 \frac{y_i - \hat{y}_i}{y_i} & \text{for } y_i \neq \text{Magic Number} \\ 0 & \text{for } y_i = \text{Magic Number} \end{cases}$$

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

```
1 def keras_model():
2     # Your keras_model define here
3     return model
4
5 model = keras_model()
6 # remember to import astroNN's loss function first
7 model.compile(loss=mean_percentage_error, ...)
```

### 1.7.9 Categorical Cross-Entropy

`astroNN.nn.losses.categorical_crossentropy(y_true, y_pred, sample_weight=None, 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
- **sample\_weight** (*Union(tf.Tensor, tf.Variable, list)*) – Sample weights
- **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  $\epsilon$  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

```

1 def keras_model():
2     # Your keras_model define here
3     return model
4
5 model = keras_model()
6 # remember to import astroNN's loss function first
7 model.compile(loss=categorical_crossentropy(from_logits=False), ...)
```

### 1.7.10 Binary Cross-Entropy

`astroNN.nn.losses.binary_crossentropy(y_true, y_pred, sample_weight=None, 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`
- **sample\_weight** (*Union(tf.Tensor, tf.Variable, list)*) – Sample weights

#### 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  $\epsilon$  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

```
1 def keras_model():
2     # Your keras_model define here
3     return model
4
5 model = keras_model()
6 # remember to import astroNN's loss function first
7 model.compile(loss=binary_crossentropy(from_logits=False), ...)
```

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

`astroNN.nn.losses.robust_categorical_crossentropy(y_true, y_pred, logit_var, sample_weight)`

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
- **sample\_weight** (`Union(tf.Tensor, tf.Variable, list)`) – Sample weights

#### 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](https://arxiv.org/abs/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](https://arxiv.org/abs/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](https://arxiv.org/abs/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

```

1 def keras_model():
2     # Your keras_model define here
3
4     # model for the training process
5     model = Model(inputs=[input_tensor], outputs=[output, variance_output])
6
7     # model for the prediction
8     model_prediction = Model(inputs=input_tensor, outputs=[output, variance_output])
9
10    variance_output = Dense(name='predictive_variance', ...)
11    output = Dense(name='output', ...)
12
13    predictive_variance_loss = bayesian_categorical_crossentropy_var_wrapper(output)
14    output_loss = bayesian_categorical_crossentropy_wrapper(predictive_variance)
15
16    return model, model_prediction, output_loss, predictive_variance_loss
17
18 model, model_prediction, output_loss, predictive_variance_loss = keras_model()
19 # remember to import astroNN loss function first
20 model.compile(loss={'output': output_loss, 'variance_output': predictive_variance_loss},
    ↪ ...)

```

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

`astroNN.nn.losses.robust_binary_crossentropy(y_true, y_pred, logit_var, sample_weight)`

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
- **sample\_weight** (`Union(tf.Tensor, tf.Variable, list)`) – Sample weights

### 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](https://arxiv.org/abs/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](https://arxiv.org/abs/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

```

1 def keras_model():
2     # Your keras_model define here
3
4     # model for the training process
5     model = Model(inputs=[input_tensor], outputs=[output, variance_output])
6
7     # model for the prediction
8     model_prediction = Model(inputs=input_tensor, outputs=[output, variance_output])
9
10    variance_output = Dense(name='predictive_variance', ...)
11    output = Dense(name='output', ...)
12
13    predictive_variance_loss = bayesian_binary_crossentropy_var_wrapper(output)
14    output_loss = bayesian_binary_crossentropy_wrapper(predictive_variance)
15
16    return model, model_prediction, output_loss, predictive_variance_loss
17
18 model, model_prediction, output_loss, predictive_variance_loss = keras_model()
19 # remember to import astroNN loss function first
20 model.compile(loss={'output': output_loss, 'variance_output': predictive_variance_loss},
    ↪ ...)

```

### 1.7.13 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

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

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

```

1 def keras_model():
2     # Your keras_model define here
3     return model
4
5 model = keras_model()
6 # remember to import astroNN's metrics function first
7 model.compile(metrics=categorical_accuracy, ...)
```

---

**Note:** Please make sure you use categorical\_accuracy when using categorical\_crossentropy as the loss function

---

### 1.7.14 Binary Classification Accuracy

astroNN.nn.losses.binary\_accuracy(\*args, \*\*kwargs)

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

#### Returns

Binary accuracy

#### Return type

tf.Tensor

#### 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

```
1 def keras_model():
2     # Your keras_model define here
3     return model
4
5 model = keras_model()
6 # remember to import astroNN's metrics function first
7 model.compile(metrics=binary_accuracy(from_logits=False), ...)
```

---

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

---

### 1.7.15 Zeros Loss

`astroNN.nn.losses.zeros_loss(y_true, y_pred, sample_weight=None)`

Always return zeros

#### Parameters

- **y\_true** (*Union(tf.Tensor, tf.Variable)*) – Ground Truth
- **y\_pred** (*Union(tf.Tensor, tf.Variable)*) – Prediction
- **sample\_weight** (*Union(tf.Tensor, tf.Variable, list)*) – Sample weights

#### 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`

```
1 def keras_model():
2     # Your keras_model define here
3     return model
4
5 model = keras_model()
6 # remember to import astroNN's loss function first
7 model.compile(loss=zeros_loss, ...)
```

## 1.8 Layers

astroNN provides some customized layers under `astroNN.nn.layers` module which built on tensorflow.keras. You can just treat astroNN customized layers as conventional Keras layers.

### 1.8.1 Monte Carlo Dropout Layer

**class** astroNN.nn.layers.MCDropout(\*args, \*\*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  $y_i$

$$r_i = \text{Bernoulli}(p)$$

$$\hat{y}_i = r_i * y_i$$

And here is an example of usage

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

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

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

## 1.8.2 Monte Carlo Dropout with Continuous Relaxation Layer Wrapper

**class** astroNN.nn.layers.MCConcreteDropout(\*args, \*\*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](https://arxiv.org/abs/1506.02557) on variational dropout. But *MCConcreteDropout* focuses on the role and importance of dropout with Bayesian technique.

And here is an example of usage

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

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

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

### 1.8.3 Monte Carlo Spatial Dropout Layer

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

**class** astroNN.nn.layers.**MCSpatialDropout1D**(\*args, \*\*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(\*args, \*\*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](https://arxiv.org/abs/1411.4280)

And here is an example of usage

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

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

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

## 1.8.4 Monte Carlo Gaussian Dropout Layer

**class** astroNN.nn.layers.MCGaussianDropout(\*args, \*\*kwargs)

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

### 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  $y_i$

$$r_i = \mathcal{N}\left(1, \sqrt{\frac{p}{1-p}}\right)$$

$$\hat{y}_i = r_i * y_i$$

And here is an example of usage

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

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

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

## 1.8.5 Monte Carlo Batch Normalization Layer

**class** astroNN.nn.layers.MCBatchNorm(\*args, \*\*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

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

And here is an example of usage

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

## 1.8.6 Error Propagation Layer

**class** astroNN.nn.layers.**ErrorProp**(\*args, \*\*kwargs)

Propagate Error Layer by adding gaussian noise (mean=0, std=err) during testing phase from input\_err tensor

**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** (*list*[*tf.Tensor*]) – a list of Tensor which [input\_tensor, input\_error\_tensor]

**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

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

And here is an example of usage

```
1 def keras_model():
2     # Your keras_model define here, assuming you are using functional API
3     input = Input(...)
4     input_with_error = ErrorProp()([input, input_error])
5     return model
```

## 1.8.7 KL-Divergence Layer for Variational Autoencoder

```
class astroNN.nn.layers.KLDivergenceLayer(*args, **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

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

And here is an example of usage

```

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

```

## 1.8.8 Polynomial Fitting Layer

**class** astroNN.nn.layers.PolyFit(\*args, \*\*kwargs)

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=[[w0]], [[w1]], ..., [[wn]]]`) 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=[[w0,1,0, w0,1,1, ..., w0,1,j], [w0,2,0, w0,2,1, ..., w0,2,j], ... [w0,i,0, w0,i,1, ..., w0,i,j]], ..., [[wn,1,0, wn,1,1, ..., wn,1,j], [wn,2,0, wn,2,1, ..., wn,2,j], ... [wn,i,0, wn,i,1, ..., wn,i,j]]]`)

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

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

*PolyFit* can be imported by

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

And here is an example of usage

```
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 tensorflow import 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]
```

(continues on next page)



(continued from previous page)

```

16 input = Input(shape=[1, ])
17 # set initial weights
18 output = PolyFit(deg=1, use_xbias=False, init_w=[[0.1]], [[-0.05]]), name='polyfit
19 ↪')(input)
20 model = Model(inputs=input, outputs=output)
21
22 # predict without training (i.e. without gradient updates)
23 np.allclose(model.predict(random_xdata), random_ydata)
24 >>> True # True means prediction approx close enough

```

### 1.8.9 Mean and Variance Calculation Layer for Bayesian Neural Net

**class** astroNN.nn.layers.FastMCInferenceMeanVar(\*args, \*\*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 want 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

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

And here is an example of usage

```
1 def keras_model():
2     # Your keras_model define here, assuming you are using functional API
3     input = Input(...)
4     monte_carlo_dropout = FastMCInference(mc_num_here)
5     # some layer here, you should use MCDropout from astroNN instead of Dropout from
6     ↪Tensorflow:)
7     result_mean_var = FastMCInferenceMeanVar()(previous_layer_here)
8     return model
9
10 model.compile(loss=loss_func_here, optimizer=optimizer_here)
11
12 # Use the model to predict
13 output = model.predict(x)
14
15 # with dropout variational inference
16 # prediction and model uncertainty (variance) from the model
17 mean = output[0]
18 variance = output[1]
```

### 1.8.10 Repeat Vector Layer for Bayesian Neural Net

```
class astroNN.nn.layers.FastMCRepeat(*args, **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 want 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

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

And here is an example of usage

```
1 def keras_model():
2     # Your keras_model define here, assuming you are using functional API
3     input = Input(...)
4     monte_carlo_dropout = FastMCRepeat(mc_num_here)
5     # some layer here, you should use MCDropout from astroNN instead of Dropout from
    ↪Tensorflow:)
6     result_mean_var = FastMCInferenceMeanVar()(previous_layer_here)
7     return model
8
9 model.compile(loss=loss_func_here, optimizer=optimizer_here)
10
11 # Use the model to predict
12 output = model.predict(x)
13
14 # with dropout variational inference
15 # prediction and model uncertainty (variance) from the model
16 mean = output[0]
17 variance = output[1]
```

### 1.8.11 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)

2021-Apr-14 - Updated - 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

```
1 from astroNN.nn.layers import FastMCInference
2
3 # keras_model is your keras model with 1 output which is a concatenation of labels_
  ↳ prediction and predictive variance
4 keras_model = Model(...)
5
6 # fast_mc_model is the new keras model capable to do fast monte carlo integration on GPU
7 fast_mc_model = FastMCInference(keras_model)
8
9 # You can just use keras API with the new model such as
10 result = fast_mc_model.predict(.....)
11
12 # here is the result dimension
13 predictions = result[:, :(result.shape[1] // 2), 0] # mean prediction
14 mc_dropout_uncertainty = result[:, :(result.shape[1] // 2), 1] * (self.labels_std ** 2)
  ↳ # model uncertainty
15 predictions_var = np.exp(result[:, (result.shape[1] // 2):, 0]) * (self.labels_std ** 2)
  ↳ # predictive uncertainty
```

### 1.8.12 Gradient Stopping Layer

**class** astroNN.nn.layers.StopGrad(\*args, \*\*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

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

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

```
1 def keras_model():
2     # Your keras_model define here, assuming you are using functional API
3     input = Input(...)
4     # some layers ...
5     stopped_grad_layer = StopGrad(...)
6     # some layers ...
7     return model
```

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

```

1 from astroNN.nn.layers import StopGrad
2 # we use zeros loss just to demonstrate StopGrad works and no error backprop from
  ↳ StopGrad layer
3 from astroNN.nn.losses import zeros_loss
4 import numpy as np
5 from astroNN.shared.nn_tools import cpu_fallback
6 from tensorflow import keras
7
8 cpu_fallback() # force tf to use CPU
9
10 Input = keras.layers.Input
11 Dense = keras.layers.Dense
12 concatenate = keras.layers.concatenate
13 Model = keras.models.Model
14
15 # Data preparation
16 random_xdata = np.random.normal(0, 1, (100, 7514))
17 random_ydata = np.random.normal(0, 1, (100, 25))
18 input2 = Input(shape=[7514])
19 dense1 = Dense(100, name='normaldense')(input2)
20 dense2 = Dense(25, name='wanted_dense')(input2)
21 dense2_stopped = StopGrad(name='stopgrad', always_on=True)(dense2)
22 output2 = Dense(25, name='wanted_dense2')(concatenate([dense1, dense2_stopped]))
23 model2 = Model(inputs=input2, outputs=[output2, dense2])
24 model2.compile(optimizer=keras.optimizers.SGD(lr=0.1),
25               loss={'wanted_dense2': 'mse', 'wanted_dense': zeros_loss})
26 weight_b4_train = model2.get_layer(name='wanted_dense').get_weights()[0]
27 weight_b4_train2 = model2.get_layer(name='normaldense').get_weights()[0]
28 model2.fit(random_xdata, [random_ydata, random_ydata])
29 weight_a4_train = model2.get_layer(name='wanted_dense').get_weights()[0]
30 weight_a4_train2 = model2.get_layer(name='normaldense').get_weights()[0]
31
32 print(np.all(weight_b4_train == weight_a4_train))
33 >>> True # meaning all the elements from Dense with StopGrad layer are equal due to no
  ↳ gradient update
34 print(np.all(weight_b4_train2 == weight_a4_train2))
35 >>> False # meaning not all the elements from normal Dense layer are equal due to
  ↳ gradient update

```

### 1.8.13 Boolean Masking Layer

**class** astroNN.nn.layers.BoolMask(\*args, \*\*kwargs)

Boolean Masking layer, please notice it is best to flatten input before using BoolMask

**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

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

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

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

## 1.8.14 TensorInput Layer

**class** astroNN.nn.layers.**TensorInput**(\*args, \*\*kwargs)

TensorInput layer

**Parameters**

**tensor** (*tf.Tensor*) – tensor, usually is a tensor generating random number

**Returns**

A layer

**Return type**

*object*

**History**

2020-May-3 - 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*

*TensorInput* takes tensorflow tensor as layer initialization and return the tensor.

*TensorInput* can be imported by

```
1 from astroNN.nn.layers import TensorInput
```

For example, if you want to generate random tensor as other layers input and do not want it to register it as model input, you can

```
1 from astroNN.nn.layers import TensorInput
2 # we use zeros loss just to demonstrate StopGrad works and no error backprop from_
  ↳ StopGrad layer
3 from astroNN.nn.losses import zeros_loss
4 import numpy as np
5 from astroNN.shared.nn_tools import cpu_fallback
6 import tensorflow as tf
7 from tensorflow import keras
8
9 cpu_fallback() # force tf to use CPU
10
11 Input = keras.layers.Input
12 Dense = keras.layers.Dense
13 concatenate = keras.layers.concatenate
14 Model = keras.models.Model
15
16 # Data preparation
17 random_xdata = np.random.normal(0, 1, (100, 7514))
18 random_ydata = np.random.normal(0, 1, (100, 25))
19 input1 = Input(shape=[7514])
20 input2 = TensorInput(tensor=tf.random.normal(mean=0., stddev=1., shape=tf.
  ↳ shape(input1)))([])
21 output = Dense(25, name='dense')(concatenate([input1, input2]))
22 model = Model(inputs=input1, outputs=output)
23 model.compile(optimizer=keras.optimizers.SGD(lr=0.1),
24               loss='mse')
25 print(model.input_names)
```

(continues on next page)



(continued from previous page)

```
26 >>> ['input_1'] # only input_1 as input_2 is not really an input we requiring user to
↪ input
```

## 1.9 Callbacks and Utilities

A callback is a set of functions under `astroNN.nn.callbacks` and `astroNN.nn.utilities` modules to be applied at given stages of the training procedure. `astroNN` provides some customized callbacks which built on `tensorflow.keras`. You can just treat `astroNN` customized callbacks as conventional Keras callbacks.

`astroNN` also contains some handy utilities for data processing

### 1.9.1 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

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

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

```
1 def keras_model():
2     # Your keras_model define here
3     return model
4
5 # Create a Virtual_CSVLogger instance first
6 csvlogger = VirutalCSVLogger()
7
8 # Default filename is training_history.csv
9 # You have to set filename first before passing to Keras
10 csvlogger.filename = 'training_history.csv'
11
12 model = keras_model()
13 model.compile(...)
14
15 model.fit(...,callbacks=[csvlogger])
16
17 # Save the file to current directory
18 csvlogger.savefile()
19
20 # OR to save the file to other directory
21 csvlogger.savefile(folder_name='some_folder')
```

## 1.9.2 Raising Error on Nan (Callback)

**class** astroNN.nn.callbacks.**ErrorOnNaN**(monitor='loss')

Callback that raise error when a NaN is encountered.

**Returns**

callback instance

**Return type**

object

**History**

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

2021-Apr-22 - 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.

## 1.9.3 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)

```

1 # If we have some data
2 data = np.array([[1,2,3], [9,8,7]])
3
4 # The normalized data, mean std will as follow by this mode
5 norm_data = array([[1,2,3], [9,8,7]])
6 # the mean and standard derivation used to do the normalization
7 mean = [0.]
8 std = [1.]

```

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

```

1 # If we have some data
2 data = np.array([[1,2,3], [9,8,7]])
3
4 # The normalized data, mean std will as follow by this mode
5 norm_data = array([[ -1.28653504, -0.96490128, -0.64326752], [ 1.28653504,  0.96490128,  0.64326752]])
6 # the mean and standard derivation used to do the normalization
7 mean = [5.0]
8 std = [3.11]

```

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

```

1 # If we have some data
2 data = np.array([[1,2,3], [9,8,7]])
3
4 # The normalized data, mean std will as follow by this mode
5 norm_data = array([[ -4., -3., -2.], [ 4.,  3.,  2.]])
6 # the mean and standard derivation used to do the normalization
7 mean = [5., 5., 5.]
8 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

```

1 # If we have some data
2 data = array([[1,2,3], [9,8,7]])
3
4 # The normalized data, mean std will as follow by this mode
5 norm_data = array([[ -1., -1., -1.], [ 1.,  1.,  1.]])
6 # the mean and standard derivation used to do the normalization
7 mean = [5., 5., 5.]
8 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

```

1 # If we have some data
2 data = np.array([[255,125,100], [99,87,250]])
3

```

(continues on next page)

(continued from previous page)

```

4 # The normalized data, mean std will as follow by this mode
5 norm_data = array([[ 1. , -0.01960784, -0.21568627], [-0.22352941, -0.31764706,  0.
   ↪ 96078431]])
6 # the mean and standard derivation used to do the normalization
7 mean = [127.5]
8 std = [127.5]

```

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

```

1 # To set the normalization mode for input and labels
2 astronn_neuralnet.input_norm_mode = ...
3 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

```

1 from astroNN.nn.utilities.normalizer import Normalizer
2 import numpy as np
3
4 # Make some data up
5 data = np.array([[1.,2.,3.], [9.,8.,7.]])
6
7 # Setup a normalizer instance with a mode, lets say mode 1
8 normer = Normalizer(mode=1)
9
10 # Use the instance method normalize to normalize the data
11 norm_data = normer.normalize(data)
12
13 print(norm_data)
14 >>> array([[ -1.28653504, -0.96490128, -0.64326752], [ 1.28653504,  0.96490128,  0.
   ↪ 64326752]])
15 print(normer.mean_labels)
16 >>> 5.0
17 print(normer.std_labels)
18 >>> 3.1091263510296048
19
20 # You can use the same instance (with same mean and std and mode) to demormalize data
21 denorm_data = normer.denormalize(data)
22
23 print(denorm_data)
24 >>> array([[1.,2.,3.], [9.,8.,7.]])

```

### 1.9.4 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)

```

1 from astroNN.nn import intpow_avx2
2 import tensorflow as tf
3
4 print(intpow_avx2(tf.constant([-1.2]), 2))
5 >>> tf.Tensor([1.44], shape=(1,), dtype=float32)
6
7 print(tf.pow(tf.constant([-1.2]), 2))
8 # if your tensorflow is compiled with AVX2 or --fast-math
9 >>> tf.Tensor([nan], shape=(1,), dtype=float32)
10 # if your tensorflow is NOT compiled with AVX2 or --fast-math
11 >>> tf.Tensor([1.44], shape=(1,), dtype=float32)

```

### 1.9.5 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.Quantity, 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.Quantity`, return without `astropy` units in all case**Parameters**

- **x** (Union[ndarray, float, `astropy.Quantity`]) – prediction
- **y** (Union[ndarray, float, `astropy.Quantity`]) – 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.Quantity`, return has no `astropy` units in all case

**Parameters**

- **x** (`Union[ndarray, float, astropy.Quantity]`) – prediction
- **y** (`Union[ndarray, float, astropy.Quantity]`) – 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.Quantity`, return without `astropy` units in all case

**Parameters**

- **x** (`Union[ndarray, float, astropy.Quantity]`) – prediction
- **y** (`Union[ndarray, float, astropy.Quantity]`) – 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 `magi cnumber` and `astropy Quantity`

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

**Parameters**

- `x` (`Union[ndarray, float, astropy.Quantity]`) – prediction
- `y` (`Union[ndarray, float, astropy.Quantity]`) – 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 `magi cnumber`

**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 `magi cnumber`

**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)

## 1.10 NeuralODE

Neural ODE (astroNN.neuralODE; Neural Ordinary Differential Equation) module provides numerical integrator implemented in Tensorflow for solutions of an ODE system, and can calculate gradient.

### 1.10.1 Numerical Integrator

astroNN implemented numerical integrator in Tensorflow

```
astroNN.neuralode.odeint.odeint(func=None, x=None, t=None, aux=None, method='dop853',
                                precision=tf.float32, *args, **kwargs)
```

To computes the numerical solution of a system of first order ordinary differential equations  $y'=f(x,y)$ . Default precision at float32.

**Parameters**

- **func** (*callable*) – function of the differential equation, usually take func([position, velocity], time) and return velocity, acceleration
- **x** (*Union([tf.Tensor, numpy.ndarray, list])*) – initial x, usually is [position, velocity]
- **t** (*Union([tf.Tensor, numpy.ndarray, list])*) – set of times at which one wants the result
- **method** (*str*) – numerical integrator to use, available integrators are ['dop853', 'rk4']
- **precision** (*type*) – float precision, tf.float32 or tf.float64
- **t** – set of times at which one wants the result

**Returns**

integrated result

**Return type**

tf.Tensor

**History**

2020-May-31 - Written - Henry Leung (University of Toronto)

An example integration an ODE for  $\sin(x)$

```
1 import time
2 import pylab as plt
3 import numpy as np
4 import tensorflow as tf
5 from astroNN.shared.nn_tools import cpu_fallback, gpu_memory_manage
6 from astroNN.neuralode import odeint
7
8 cpu_fallback()
9 gpu_memory_manage()
```

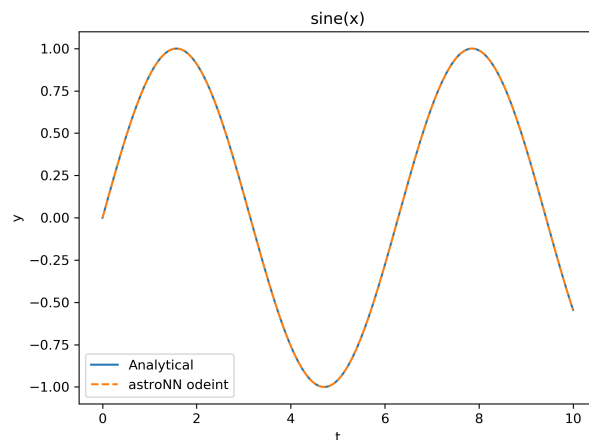
(continues on next page)

(continued from previous page)

```

10
11 # time array
12 t = tf.constant(np.linspace(0, 100, 10000))
13 # initial condition
14 true_y0 = tf.constant([0., 1.])
15 # analytical ODE system for sine wave [x, t] -> [v, a]
16 ode_func = lambda y, t: tf.stack([tf.cos(t), tf.sin(t)])
17
18 start_t = time.time()
19 true_y = odeint(ode_func, true_y0, t, method='dop853')
20 print(time.time() - start_t) # approx. 4.3 seconds on i7-9750H GTX1650
21
22 # plot the solution and compare
23 plt.figure(dpi=300)
24 plt.title("sine(x)")
25 plt.plot(t, np.sin(t), label='Analytical')
26 plt.plot(t, true_y[:, 0], ls='--', label='astroNN odeint')
27 plt.legend(loc='best')
28 plt.xlabel("t")
29 plt.ylabel("y")
30 plt.show()

```



Moreover odeint supports numerically integration in parallel, the example below integration the  $\sin(x)$  for 50 initial conditions. You can see the execution time is the same!!

```

1 start_t = time.time()
2 # initial conditions, 50 of them instead of a single initial condition
3 true_y0sss = tf.random.normal((50, 2), 0, 1)
4 # time array, 50 of them instead of the same time array for every initial condition
5 tsss = tf.random.normal((50, 10000), 0, 1)
6 true_y = odeint(ode_func, true_y0sss, tsss, method='dop853')
7 print(time.time() - start_t) # also approx. 4.3 seconds on i7-9750H GTX1650

```

## 1.10.2 Neural Network model with Numerical Integrator

You can use odeint along with neural network model, below is an example

```

1 import numpy as np
2 import tensorflow as tf
3 from astroNN.shared.nn_tools import gpu_memory_manage, cpu_fallback
4 from astroNN.neuralode import odeint
5
6 cpu_fallback()
7 gpu_memory_manage()
8
9 t = tf.constant(np.linspace(0, 1, 20))
10 # initial condition
11 true_y0 = tf.constant([0., 1.])
12
13 class MyModel(tf.keras.Model):
14     def __init__(self):
15         super(MyModel, self).__init__()
16         self.dense1 = tf.keras.layers.Dense(2, activation=tf.nn.relu)
17         self.dense2 = tf.keras.layers.Dense(16, activation=tf.nn.relu)
18         self.dense3 = tf.keras.layers.Dense(2)
19
20     def call(self, inputs, t, *args):
21         inputs = tf.expand_dims(inputs, axis=0)
22         x = self.dense2(self.dense1(inputs))
23         return tf.squeeze(self.dense3(x))
24
25 model = MyModel()
26
27 with tf.GradientTape() as g:
28     g.watch(true_y0)
29     y = odeint(model, true_y0, t)
30 # gradient of the result w.r.t. model's weights
31 g.gradient(y, model.trainable_variables) # well define, no None, no inf or no NaN

```

## 1.11 Neural Nets Classes and Basic Usage

### 1.11.1 Available astroNN Neural Net Classes

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

```

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

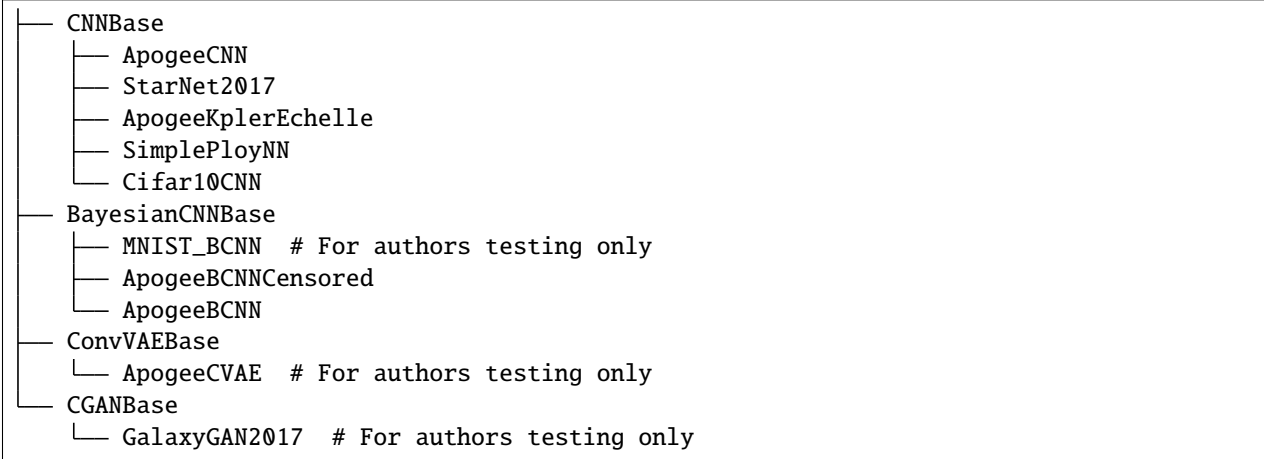
GeneratorMaster (astroNN.nn.utilities.generator.GeneratorMaster)
├── CNNDDataGenerator
├── Bayesian_DataGenerator
├── CVAE_DataGenerator

NeuralNetMaster (astroNN.models.base_master_nn.NeuralNetMaster)

```

(continues on next page)

(continued from previous page)



### 1.11.2 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 actual 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)

#### **property 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)**

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

**Returns**

An array of Hessian

**Return type**

*ndarray*

**History**

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

**property 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)

### **property 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 or path*) – Folder name/path 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)

**transfer\_weights(model, exclusion\_output=False)**

Transfer weight of a model to current model if possible # TODO: remove layers after successful transfer so wont mix up?

**Parameters**

- **model** (*astroNN.model.NeuralNetMaster* or *keras.models.Model*) – astroNN model
- **exclusion\_output** (*bool*) – whether to exclude output in the transfer or not

**Returns**

bool

**History**

2022-Mar-06 - Written - Henry Leung (University of Toronto)

**property 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)

**fit**(*input\_data*, *labels*, *sample\_weight=None*)

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
- **sample\_weight** (*Union([NoneType, ndarray])*) – Sample weights (if any)

**Returns**

None

**Return type**

NoneType

**History**

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

**fit\_on\_batch**(*input\_data*, *labels*, *sample\_weight=None*)

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
- **sample\_weight** (*Union([NoneType, ndarray])*) – Sample weights (if any)

**Returns**

None

**Return type**

NoneType

**History**

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

**predict**(*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**

*ndarray*

**History**

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

**recompile**(*loss=None, weighted\_metrics=None, loss\_weights=None, sample\_weight\_mode=None*)

To be used when you need to recompile a already existing model

**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)

**custom\_train\_step**(*data*)

Custom training logic

**Parameters**

**data** –

**Returns**

**evaluate**(*input\_data, labels, inputs\_err=None, labels\_err=None, batch\_size=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)

**fit**(*input\_data, labels, inputs\_err=None, labels\_err=None, sample\_weight=None, experimental=False*)

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)

- **sample\_weight** (*Union*([NoneType, ndarray])) – Sample weights (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)

**fit\_on\_batch**(*input\_data*, *labels*, *inputs\_err=None*, *labels\_err=None*, *sample\_weight=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)
- **sample\_weight** (*Union*([NoneType, ndarray])) – Sample weights (if any)

**Returns**

None

**Return type**

NoneType

**History**

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

**predict**(*input\_data*, *inputs\_err=None*, *batch\_size=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)

**recompile**(*weighted\_metrics=None*, *loss\_weights=None*, *sample\_weight\_mode=None*)

To be used when you need to recompile a already existing model

## 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)

**custom\_train\_step**(*data*)

Custom training logic

### Parameters

**data** –

### Returns

**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)

**fit**(*input\_data*, *input\_recon\_target*, *sample\_weight=None*)

Train a Convolutional Autoencoder

### Parameters

- **input\_data** (*ndarray*) – Data to be trained with neural network
- **input\_recon\_target** (*ndarray*) – Data to be reconstructed
- **sample\_weight** (*Union([NoneType, ndarray])*) – Sample weights (if any)

### Returns

`None`

### Return type

`NoneType`

### History

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

**fit\_on\_batch**(*input\_data*, *input\_recon\_target*, *sample\_weight=None*)

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
- **sample\_weight** (*Union([NoneType, ndarray])*) – Sample weights (if any)

**Returns**

None

**Return type**

NoneType

**History**

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

**jacobian\_latent**(*x=None, mean\_output=False, mc\_num=1, denormalize=False*)

Calculate jacobian of gradient of latent space 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)

**predict**(*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**

ndarray

**History**

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

**predict\_decoder(*z*)**

Use the decoder to get the hidden layer encoding/representation

**Parameters**

**z** (*ndarray*) – Latent space vectors

**Returns**

output reconstruction

**Return type**

*ndarray*

**History**

2022-Dec-08 - Written - Henry Leung (University of Toronto)

**predict\_encoder(*input\_data*)**

Use the encoder to get the hidden layer encoding/representation

**Parameters**

**input\_data** (*ndarray*) – Data to be inferred with neural network

**Returns**

hidden layer encoding/representation mean and std

**Return type**

*ndarray*

**History**

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

**recompile(*loss=None, weighted\_metrics=None, loss\_weights=None, sample\_weight\_mode=None*)**

To be used when you need to recompile a already existing model

### 1.11.3 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,

```
1 # import the neural net class from astroNN first
2 from astroNN.models import ApogeeCNN
3
4 # astronnet_neuralnet is an astroNN's neural network instance
5 # In this case, it is an instance of ApogeeCNN
6 astronnet_neuralnet = ApogeeCNN()
```

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

```
1 # Just an example, if the training data is Teff,logg, Fe and absmag
2 astronnet_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

```

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

```

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

```

1 # To enable autosave
2 astronn_neuralnet.autosave = True
3
4 # To save all the stuffs, model_plot=True to plot models too, otherwise wont plot, needs ↵
5 ↵pydot_ng and graphviz
6 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

```

1 # astronn_neuralnet is an astroNN's neural network instance
2 astronn_neuralnet.input_norm_mode=0
3 astronn_neuralnet.labels_norm_mode = 0

```

You can add a list of Keras/astroNN callback by

```

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

```

So now everything is set up for training

```

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

```

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

```

1 # To save all the stuffs, model_plot=True to plot models too, otherwise wont plot, needs ↵
2 ↵pydot_ng and graphviz
3 astronn_neuralnet.save(model_plot=False)

```

### 1.11.4 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

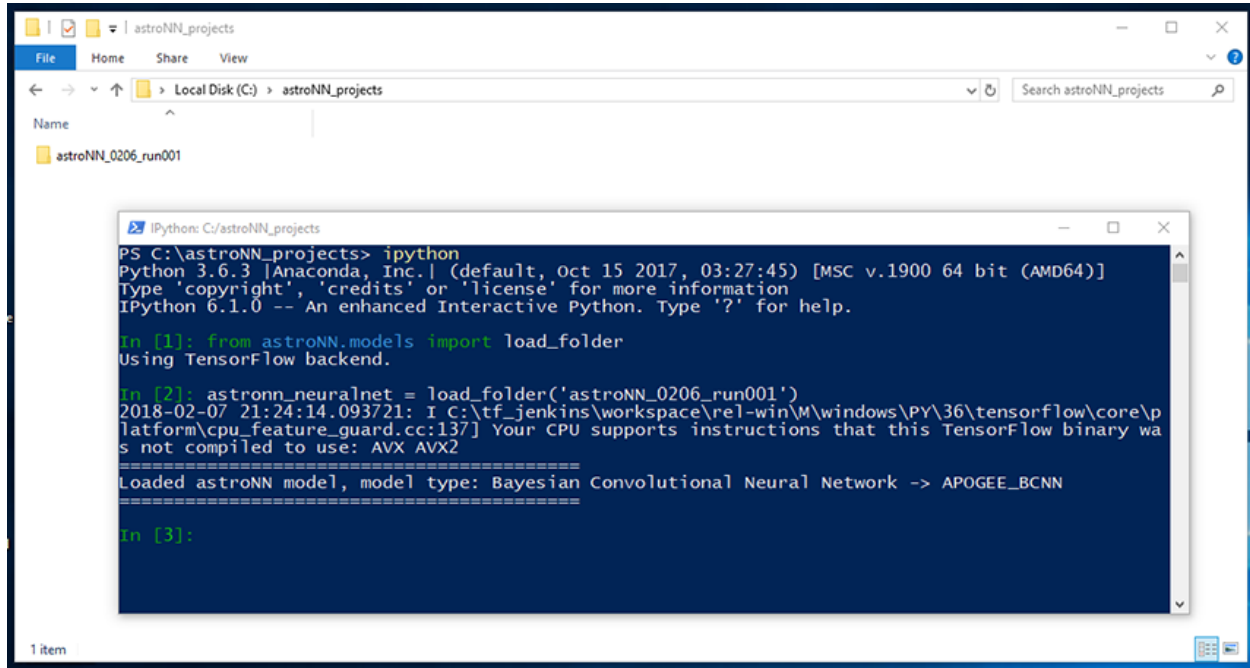
#### Return type

astroNN.nn.NeuralNetMaster.NeuralNetMaster

## History

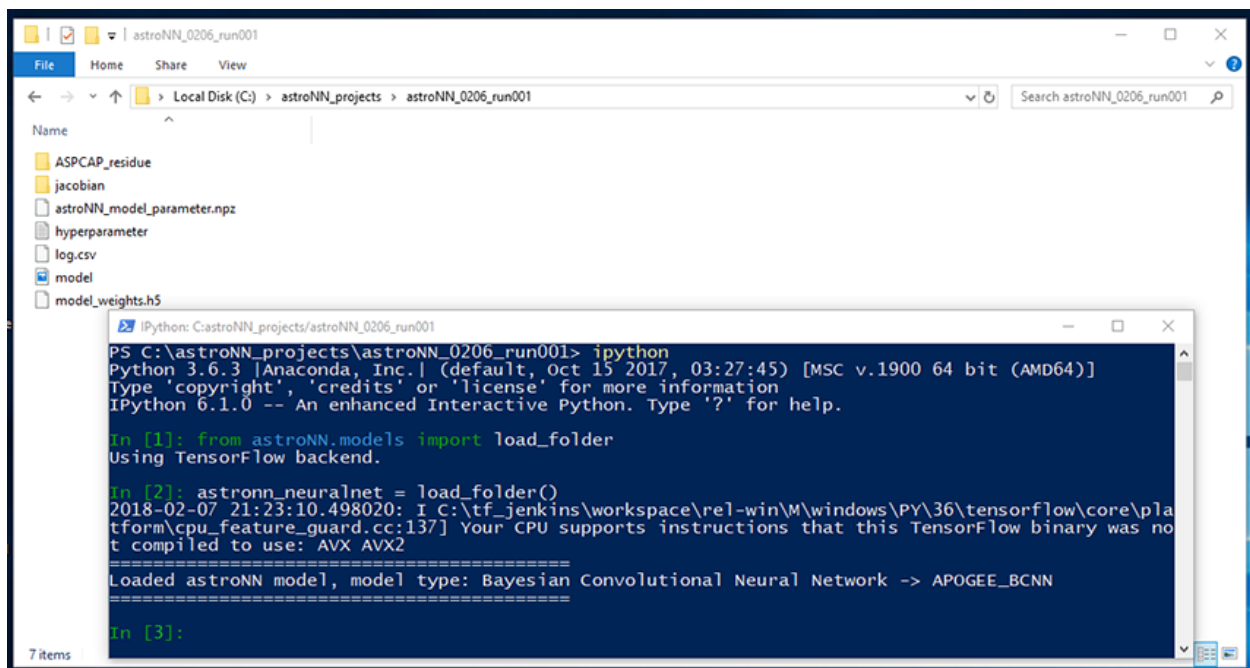
2017-Dec-29 - Written - Henry Leung (University of Toronto)

```
1 from astroNN.models import load_folder
2 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

```
1 from astroNN.models import load_folder
2 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,

```

1  # The model summary from Keras
2  astronn_neuralnet.keras_model.summary()
3
4  # The model input
5  astronn_neuralnet.keras_model.input
6
7  # The model input shape expectation
8  astronn_neuralnet.keras_model.input_shape
9
10 # The model output
11 astronn_neuralnet.keras_model.output
12
13 # The model output shape expectation
14 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)

```

1  # The targetname corresponding to output neurone
2  astronn_neuralnet.targetname
3
4  # The model input
5  astronn_neuralnet.keras_model.input
6
7  # The mean used to normalized training data
8  astronn_neuralnet.input_mean_norm
9
10 # The standard derivation used to normalized training data
11 astronn_neuralnet.input_std_norm
12
13 # The mean used to normalized training labels
14 astronn_neuralnet.labels_mean_norm
15
16 # The standard derivation used to normalized training labels
17 astronn_neuralnet.labels_std_norm

```

### 1.11.5 Load and Use Multiple astroNN Generated Folders

**Note:** astroNN fully supports eager execution now and you no longer need to context manage graph and session in order to use multiple model at the same time

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 separate Graph and Session for each astroNN neural network model. You can do:

```
1 from astroNN.models import load_folder
2
3 astronn_model_1 = load_folder("astronn_model_1")
4 astronn_model_2 = load_folder("astronn_model_2")
5 astronn_model_3 = load_folder("astronn_model_3")
6
7 with astronn_model_1.graph.as_default():
8     with astronn_model_1.session.as_default():
9         # do stuff with astronn_model_1 here
10
11 with astronn_model_2.graph.as_default():
12     with astronn_model_2.session.as_default():
13         # do stuff with astronn_model_2 here
14
15 with astronn_model_3.graph.as_default():
16     with astronn_model_3.session.as_default():
17         # do stuff with astronn_model_3 here
18
19 # For example do things with astronn_model_1 again
20 with astronn_model_1.graph.as_default():
21     with astronn_model_1.session.as_default():
22         # do more stuff with astronn_model_1 here
```

### 1.11.6 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.

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

You can always train on new data based on existing weights

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

### 1.11.7 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.

```

1  # import everything we need
2  from tensorflow import keras
3  # this is the astroNN neural net abstract class we will going to inherit from
4  from astroNN.models.CNNBase import CNNBase
5
6  regularizers = keras.regularizers
7  MaxPooling2D, Conv2D, Dense, Flatten, Activation, Input = keras.layers.MaxPooling2D, \
8  ↪keras.layers.Conv2D, \
9  ↪keras.layers.Flatten, \
10 ↪keras.layers.Dense, keras.
11 ↪keras.layers.Activation, keras.
12
13 # now we are creating a custom model based on astroNN neural net abstract class
14 class my_custom_model(CNNBase):
15     def __init__(self, lr=0.005):
16         # standard super for inheriting abstract class
17         super().__init__()
18
19         # some default hyperparameters
20         self._implementation_version = '1.0'
21         self.initializer = 'he_normal'
22         self.activation = 'relu'
23         self.num_filters = [8]
24         self.filter_len = (3, 3)
25         self.pool_length = (4, 4)
26         self.num_hidden = [128]
27         self.max_epochs = 1
28         self.lr = lr
29         self.reduce_lr_epsilon = 0.00005
30
31         self.task = 'classification'
32         # you should set the targetname some that you know what those output neurones
33 ↪are representing
34         # in this case the outpu the neurones are simply representing digits
35         self.targetname = ['Zero', 'One', 'Two', 'Three', 'Four', 'Five', 'Six', 'Seven',
36 ↪'Eight', 'Nine']
37
38         # set default input norm mode to 255 to normalize images correctly
39         self.input_norm_mode = 255
40         # set default labels norm mode to 0 (equivalent to do nothing) to normalize
41 ↪labels correctly
42         self.labels_norm_mode = 0
43
44     def model(self):
45         input_tensor = Input(shape=self._input_shape, name='input')
46         cnn_layer_1 = Conv2D(kernel_initializer=self.initializer, padding="same",
47 ↪filters=self.num_filters[0],
48 ↪kernel_size=self.filter_len)(input_tensor)
49         activation_1 = Activation(activation=self.activation)(cnn_layer_1)
50         maxpool_1 = MaxPooling2D(pool_size=self.pool_length)(activation_1)
51         flattener = Flatten()(maxpool_1)
52         layer_2 = Dense(units=self.num_hidden[0], kernel_initializer=self.

```

(continues on next page)

(continued from previous page)

```

47     ↪initializer)(flattener)
         activation_2 = Activation(activation=self.activation)(layer_2)
48         layer_3 = Dense(units=self.labels_shape, kernel_initializer=self.
         ↪initializer)(activation_2)
49         output = Activation(activation=self._last_layer_activation, name='output')(layer_
         ↪3)
50
51         model = Model(inputs=input_tensor, outputs=output)
52
53         return model

```

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

```

1  # import everything we need
2  from custom_models import my_custom_model
3  from keras.datasets import mnist
4  from keras import utils
5
6  # load MNIST
7  (x_train, y_train), (x_test, y_test) = mnist.load_data()
8  # convert to approach type
9  x_train = x_train.astype('float32')
10 x_test = x_test.astype('float32')
11 y_train = utils.to_categorical(y_train, 10)
12
13 # create a neural network instance
14 net = my_custom_model()
15
16 # train
17 net.train(x_train, y_train)
18
19 # save the model after training
20 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

```

1  # import everything we need
2  from astroNN.models import load_folder
3
4  net = load_folder()

```

OR outside the folder `trained_models_folder`

```

1  # import everything we need
2  from astroNN.models import load_folder
3
4  net = load_folder("trained_models_folder")

```

### 1.11.8 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).

## 1.12 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.

### 1.12.1 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  $P(B)$  are probability of observing A and B independently of each other.

The Bayesian interpretation of a probability 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 (similarity to estimation of  $\pi$  by MC sampling)

### 1.12.2 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

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

### 1.12.3 How is uncertainty calculated from neural network for regression task

Prediction = 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

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

For more detail, please see my demonstration [here](#)

### 1.12.4 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 tell 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.

## 1.13 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](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

### 1.13.1 (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/](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

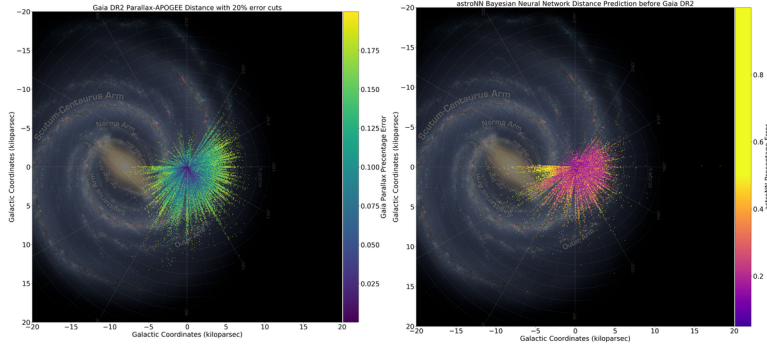
**Gaia DR1, Anderson2017 with 20% error cuts in APOGEE DR14 crossed matched:**

- 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

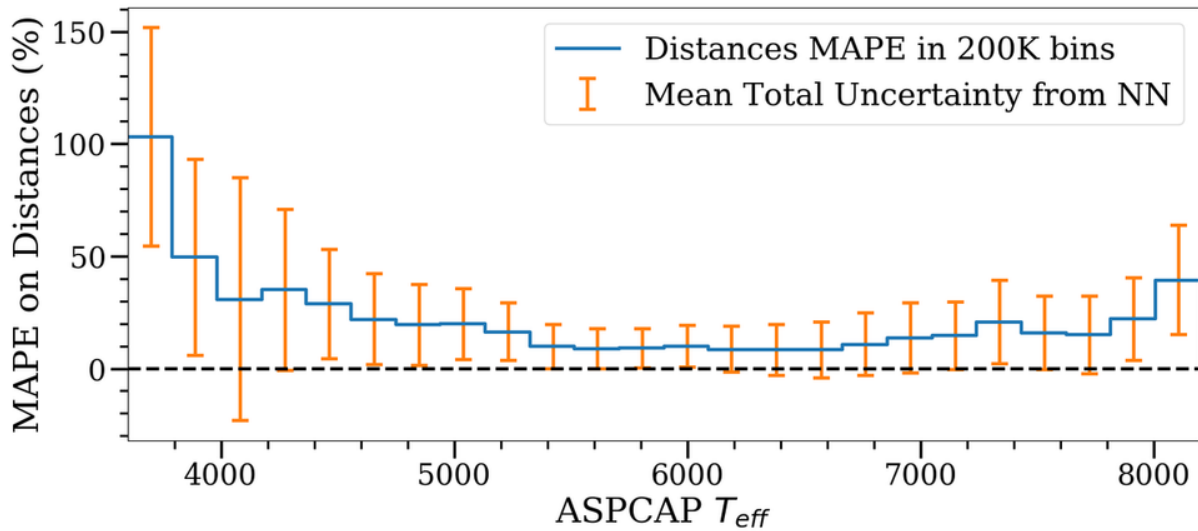
**Apogee Red Clumps - astroNN - Gaia DR2 crossed matched, Red Clumps Catalog DR14 is better than NN:**

- 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

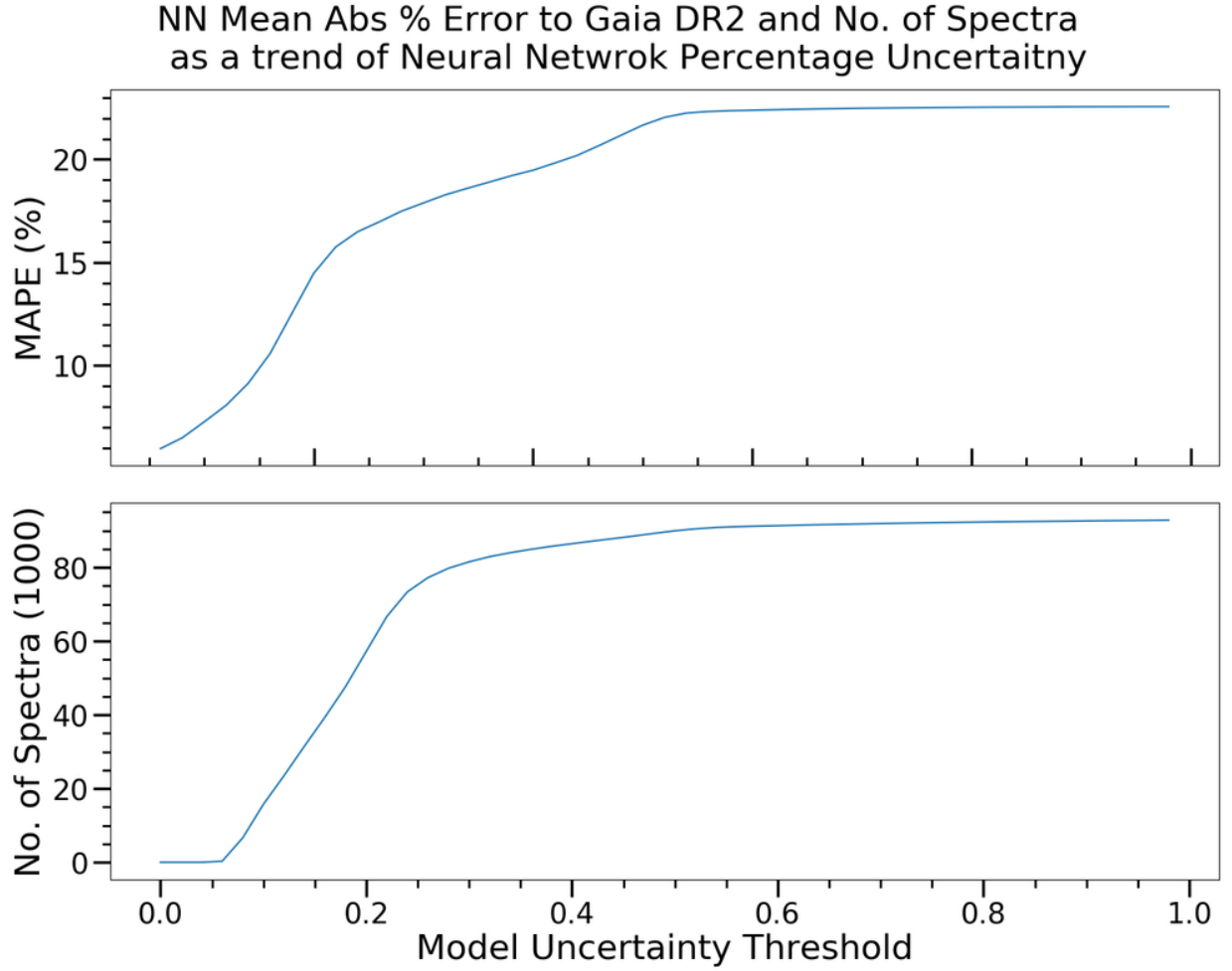


**Neural Network Mean Absolute Percentage Error to Gaia DR2 as a function of Teff**



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



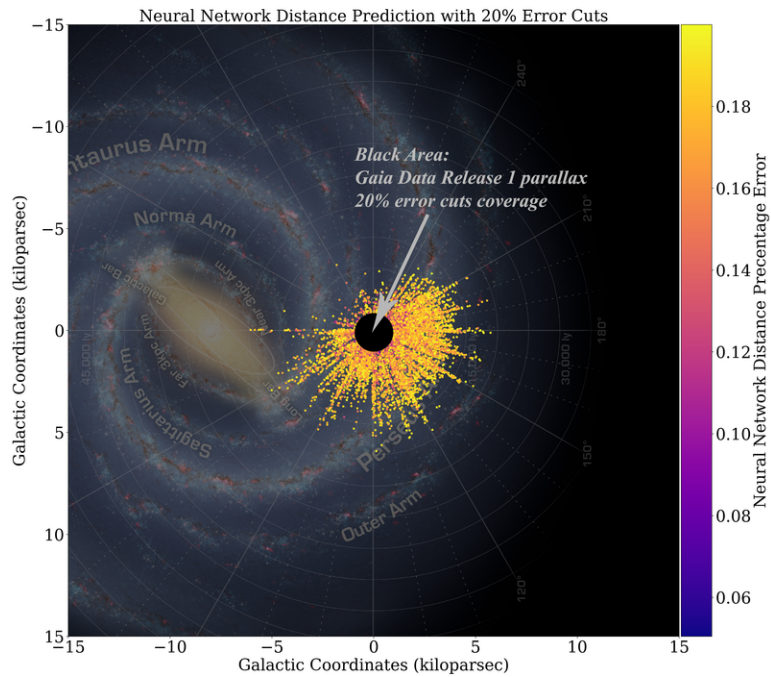


### 1.13.2 Plans/Questions

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?

### 1.13.3 Neural 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

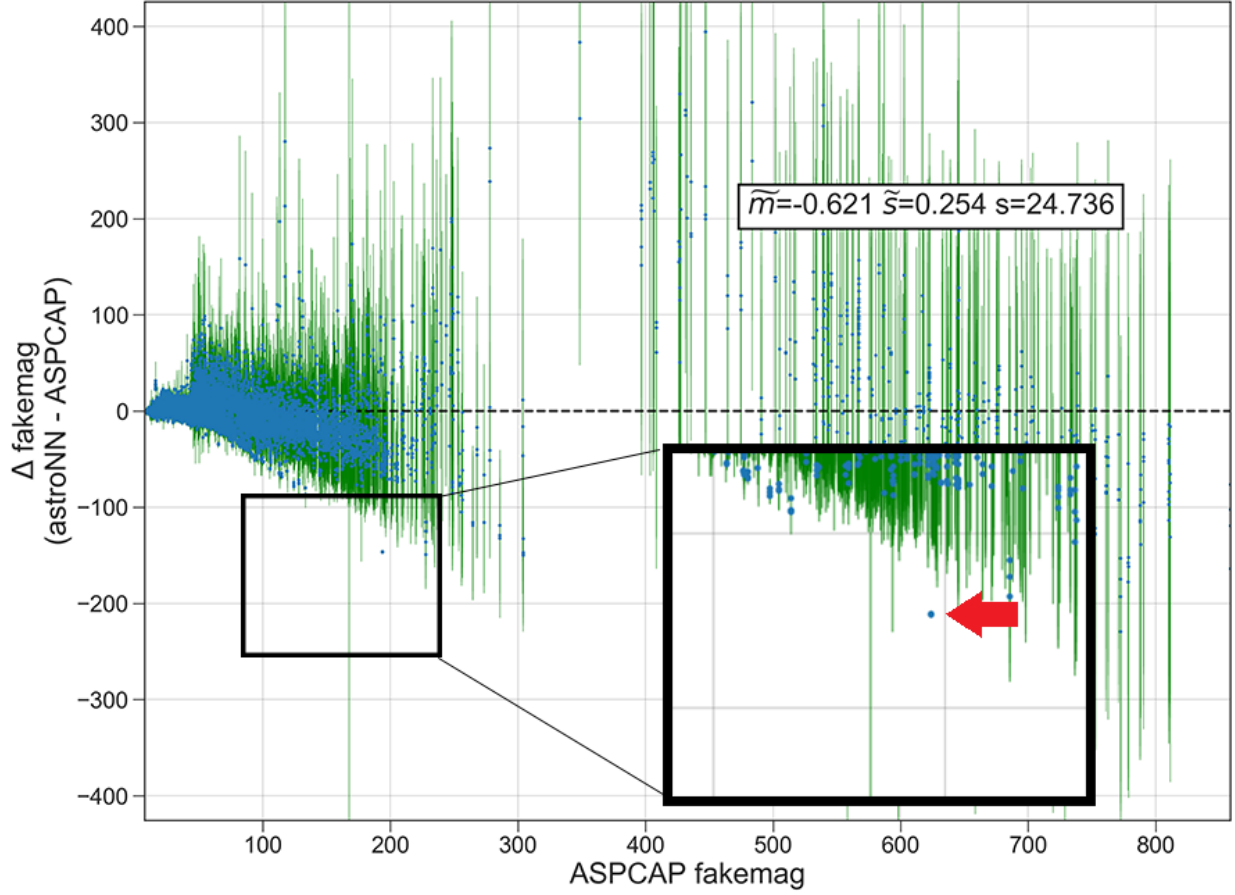


astroNN: <https://github.com/henrysky/astroNN>

Henry Leung/Jo Bovy 2018

### 1.13.4 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<sup>1</sup> : 2287.61 parsec  $\pm$  107.27 parsec
2. APOGEE\_distances BPG\_dist50<sup>2</sup> : 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

<sup>1</sup> Trained on ASPCAP parameters [Teff, Log(g) and 22 abundances] and Anderson2017 parallax

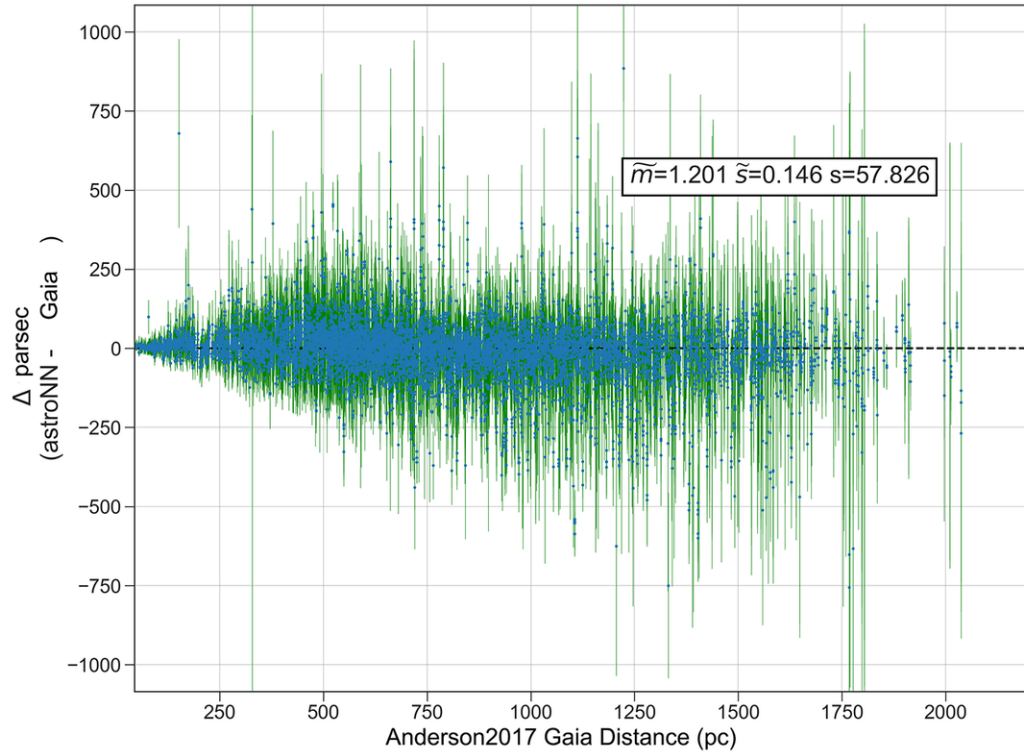
<sup>2</sup> [http://www.sdss.org/dr14/data\\_access/value-added-catalogs/?vac\\_id=apogee-dr14-based-distance-estimations](http://www.sdss.org/dr14/data_access/value-added-catalogs/?vac_id=apogee-dr14-based-distance-estimations)

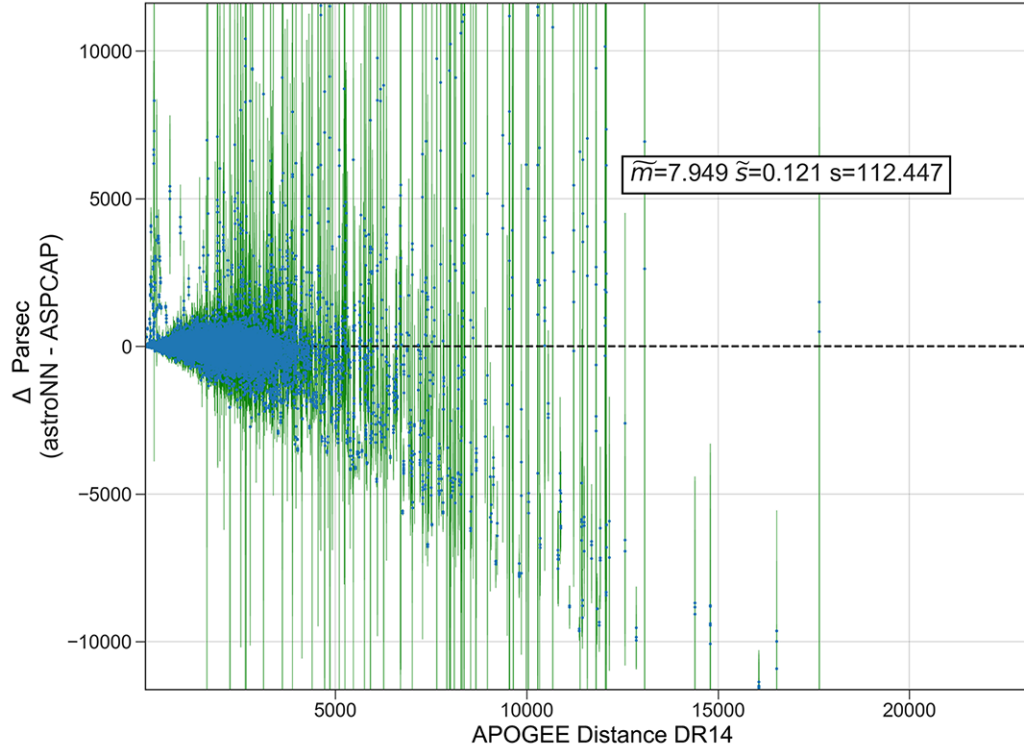
### 1.13.5 Distance Prediction with APOGEE-North Spectra

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. 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

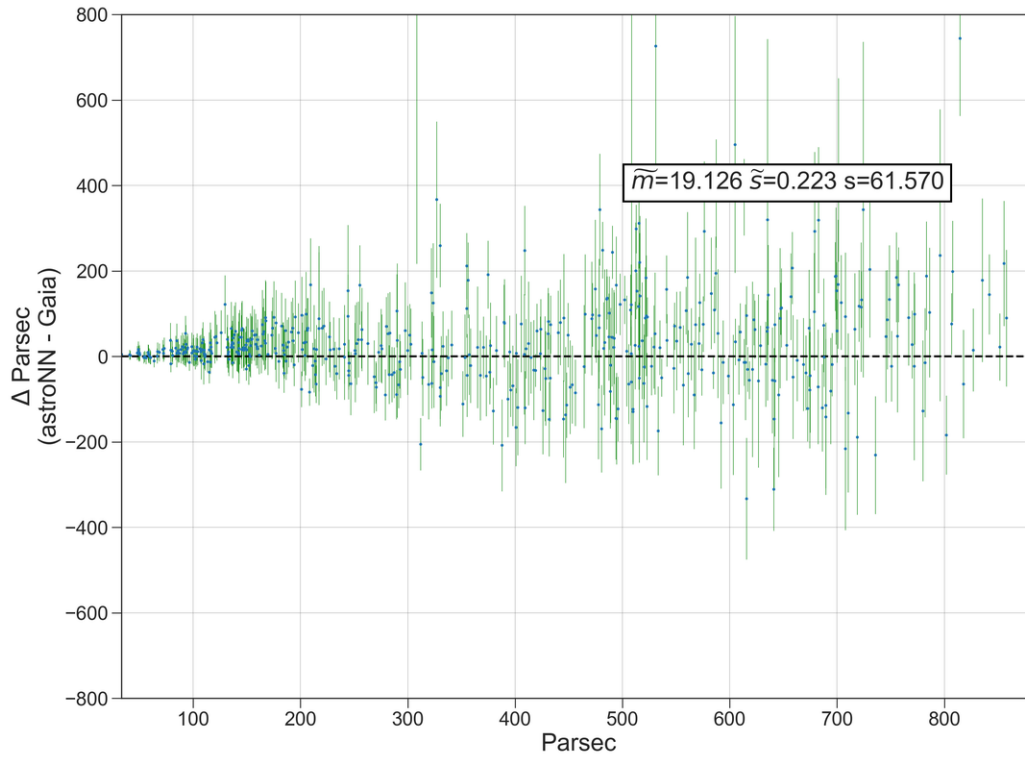




### 1.13.6 Distance Prediction with APOGEE-South Spectra

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)

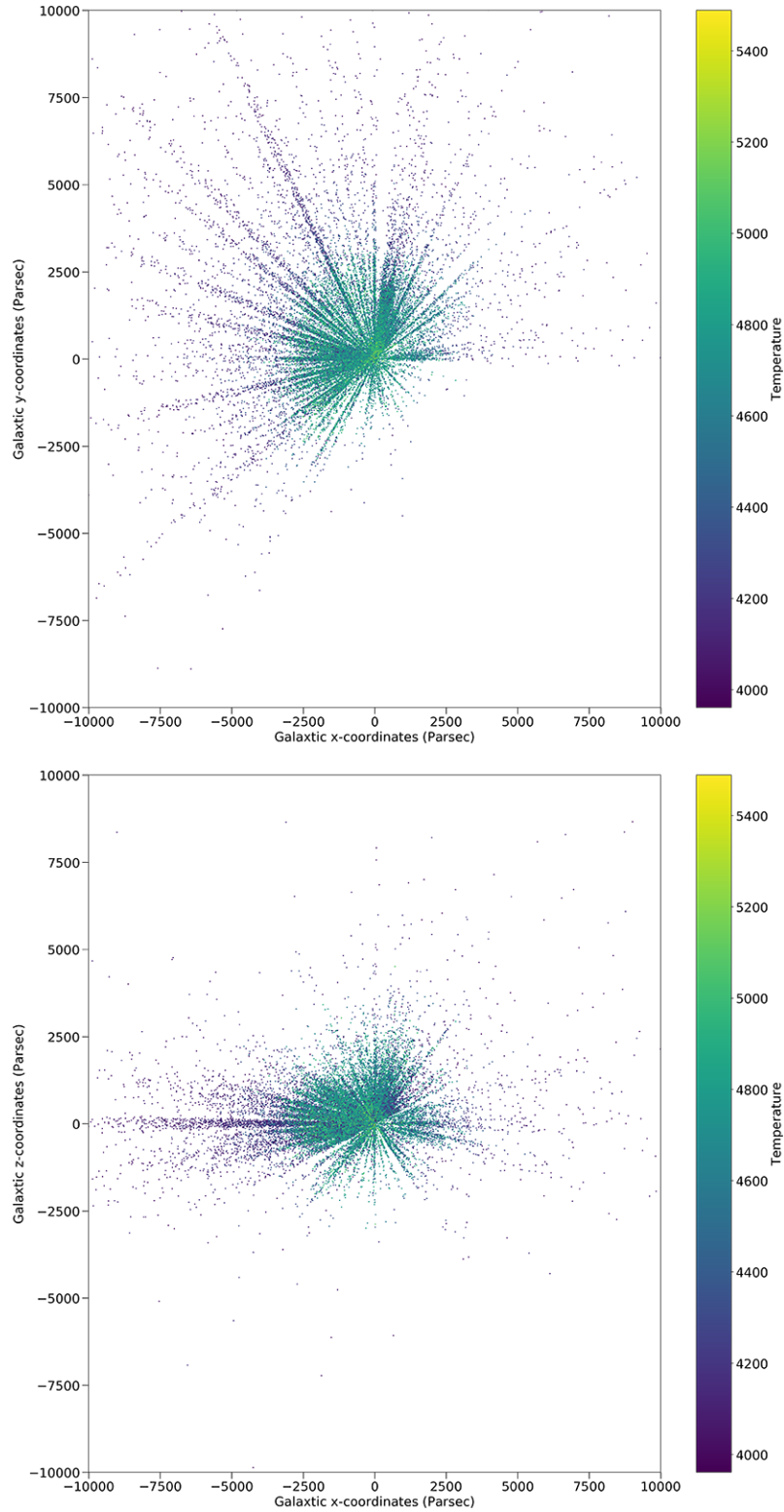


### 1.13.7 Milkyway via the Eye of Neural Network

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

## 1.14 Mini Tools for APOGEE data

---

**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](#)

---

astroNN.apogee module has a handful of tool to deal with APOGEE data. The APO Galactic Evolution Experiment employs 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 is surveying red giant stars across the full range of the Galactic bulge, bar, disk, and halo. APOGEE 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-2/>

### 1.14.1 Continuum Normalization of APOGEE Spectra

You can access the default astroNN continuum mask fro APOGEE spectra by

```
1 import os
2 import astroNN
3 import numpy as np
4
5 dr = 14
6
7 dir = os.path.join(os.path.dirname(astroNN.__path__[0]), 'astroNN', 'data', f'dr{dr}_
8 ↪contmask.npy')
9 cont_mask = np.load(dir)
```

When you do continuum normalization using astroNN, you can just use `cont_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**

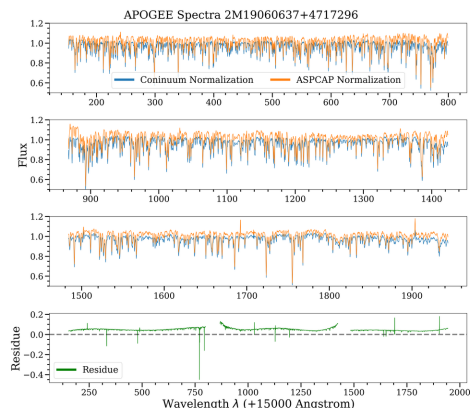
2018-Mar-21 - Written - Henry Leung (University of Toronto)

```

1 from astroNN.apogee import apogee_continuum
2
3 # spectra_errs refers to the 1-sigma error array provided by APOGEE
4 # spectra can be multiple spectra at a time
5 norm_spec, norm_spec_err = apogee_continuum(apogee_spectra, spectra_errs, cont_mask=None,
6     ↪ deg=2, dr=14)
7
8 # If you deal with bitmask too and want to set some target bits to zero, you can add
9     ↪ additional argument in apogee_continuum()
10 # You target_bit=[a list of number] or target_bit=None to use default target_bit
11 apogee_continuum(apogee_spectra, spectra_errs, cont_mask=None, deg=2, dr=14,
12     ↪ bitmask=apogee_bitmask, target_bit=None)

```

*norm\_spec* refers to the normalized spectra while *norm\_spec\_err* refers to the normalized spectra error



You can use `continuum()` to normalize any spectra while `apogee_continuum()` is specifically designed for APOGEE spectra.

```

1 from astroNN.apogee import continuum
2
3 spec, spec_err = continuum(spectra, spectra_errs, cont_mask, deg=2)

```

## 1.14.2 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)

```
1 from astroNN.apogee import chips_pix_info
2
3 info = chips_pix_info(dr=14)
4
5 # info[0] refers to the location where blue chips starts
6 # info[1] refers to the location where blue chips ends
7 # info[2] refers to the location where green chips starts
8 # info[3] refers to the location where blue chips end
9 # info[4] refers to the location where red chips starts
10 # info[5] refers to the location where red chips ends
11 # 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

```

1 from astroNN.apogee import wavelength_solution
2
3 lambda_blue, lambda_green, lambda_red = wavelength_solution(dr=16)
4
5 # lambda_blue refers to the wavelength solution for each pixel in blue chips
6 # lambda_green refers to the wavelength solution for each pixel in green chips
7 # 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

```
1 from astroNN.apogee import gap_delete
2
3 # original_spectra can be multiple spectra at a time
4 gap_deleted_spectra = gap_delete(original_spectra, dr=16)
```

## 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

```
1 from astroNN.apogee import chips_split
2
3 # original_spectra can be multiple spectra at a time
4 spectra_blue, spectra_green, spectra_red = chips_split(original_spectra, dr=16)
```

`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/dr16/algorithms/bitmasks/#collapseAPOGEE\\_PIXMASK](http://www.sdss.org/dr16/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:

```

1 from astroNN.apogee import bitmask_boolean
2 import numpy as np
3
4 spectra_bitmask = np.array([2048, 128, 1024, 512, 16, 8192, 4096, 64, 2, 32, 256, 8, 4,
5                             ↪16896])
6 boolean_output = bitmask_boolean(spectra_bitmask, target_bit=[0,1,2,3,4,5,6,7,9,12])
7 print(boolean_output)
>>> array([[False, True, False, True, True, False, True, True, True, True, False, True,
8           ↪True, True]])

```

**Decompose APOGEE Bitmask into Constitute Bits**

You can turn a APOGEE PIXMASK bit into its constitute bits

Bitmask: [http://www.sdss.org/dr16/algorithms/bitmasks/#collapseAPOGEE\\_PIXMASK](http://www.sdss.org/dr16/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)

```

1 from astroNN.apogee import bitmask_decompositor
2
3 decomposed_bits = bitmask_decompositor(single_bitmask)

```

Example:

```

1 from astroNN.apogee import bitmask_decompositor
2
3 # Create a simulated bit number
4 # Lets say this pixel is marked as 0, 5, 13 and 14 bit
5 bitmask = 2**0 + 2**5 + 2**13 + 2**14
6
7 decomposed_bits = bitmask_decompositor(bitmask)
8 # The function returns the set of original bits
9 >>> 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)

```
1 from astroNN.apogee import aspcap_mask
2
3 mask = aspcap_mask('Mg') # for example you want to get ASPCAP Mg mask
```

## 1.14.3 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/>

```
1 from astropy.io import fits
2
3 data = fits.open(local_path_to_file)
```

**allstar file**

Data Model: [https://data.sdss.org/datamodel/files/APOGEE\\_ASPCAP/APRED\\_VERS/ASPCAP\\_VERS/allStar.html](https://data.sdss.org/datamodel/files/APOGEE_ASPCAP/APRED_VERS/ASPCAP_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)

```
1 from astroNN.apogee import allstar
2
3 local_path_to_file = allstar(dr=16)
```

**allvisit file**

Data Model: [https://data.sdss.org/datamodel/files/APOGEE\\_ASPCAP/APRED\\_VERS/ASPCAP\\_VERS/allVisit.html](https://data.sdss.org/datamodel/files/APOGEE_ASPCAP/APRED_VERS/ASPCAP_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)

```
1 from astroNN.apogee import allvisit
2
3 local_path_to_file = allvisit(dr=16)
```

## Combined Spectra (aspcapStar)

Data Model: [https://data.sdss.org/datamodel/files/APOGEE\\_ASPCAP/APRED\\_VERS/ASPCAP\\_VERS/TELESCOPE/FIELD/aspcapStar.html](https://data.sdss.org/datamodel/files/APOGEE_ASPCAP/APRED_VERS/ASPCAP_VERS/TELESCOPE/FIELD/aspcapStar.html)

`astroNN.apogee.combined_spectra(dr=None, location=None, field=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]
- **field** (*str*) – Field [Optional]
- **apogee** (*str*) – Apogee ID
- **telescope** (*str*) – Telescope ID, for example ‘apo25m’ or ‘lco25m’
- **verbose** (*int*) – verbose, set 0 to silent most logging
- **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)

```
1 from astroNN.apogee import combined_spectra
2
3 local_path_to_file = combined_spectra(dr=16, location=a_location_id, apogee=a_apogee_id)
```

## Visit Spectra (apStar)

Data Model: [https://data.sdss.org/datamodel/files/APOGEE\\_REDUX/APRED\\_VERS/stars/TELESCOPE/FIELD/apStar.html](https://data.sdss.org/datamodel/files/APOGEE_REDUX/APRED_VERS/stars/TELESCOPE/FIELD/apStar.html)

`astroNN.apogee.visit_spectra(dr=None, location=None, field=None, apogee=None, telescope=None, verbose=1, flag=None, commission=False)`

Download the required individual spectra file a.k.a apStar or asStar

### Parameters

- **dr** (*int*) – APOGEE DR
- **location** (*int*) – Location ID [Optional]
- **field** (*str*) – Field [Optional]
- **apogee** (*str*) – Apogee ID
- **telescope** (*str*) – Telescope ID, for example ‘apo25m’ or ‘lco25m’
- **verbose** (*int*) – verbose, set 0 to silent most logging



- **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)

```
1 from astroNN.apogee import visit_spectra
2
3 local_path_to_file = visit_spectra(dr=16, location=a_location_id, apogee=a_apogee_id)
```

**astroNN catalogue for APOGEE**

Introduction: [https://www.sdss.org/dr16/data\\_access/value-added-catalogs/?vac\\_id=the-astroNN-catalog-of-abundances,-distances,-and-ages-for-apogee-dr16-stars](https://www.sdss.org/dr16/data_access/value-added-catalogs/?vac_id=the-astroNN-catalog-of-abundances,-distances,-and-ages-for-apogee-dr16-stars)

Data Model (DR16): [https://data.sdss.org/datamodel/files/APOGEE\\_ASTRONN/apogee\\_astroNN.html](https://data.sdss.org/datamodel/files/APOGEE_ASTRONN/apogee_astroNN.html)

`astroNN.apogee.downloader.apogee_astroNN(dr=None, flag=None)`

**Download the apogee\_astroNN file (catalog of astroNN stellar parameters, abundances, distances and orbital parameters 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**

2019-Dec-10 - Written - Henry Leung (University of Toronto)

```
1 from astroNN.apogee import apogee_astroNN
2
3 local_path_to_file = apogee_astroNN(dr=16)
```

## Red Clumps of SDSS Value Added Catalogs

Introduction: [http://www.sdss.org/dr16/data\\_access/value-added-catalogs/?vac\\_id=apogee-red-clump-rc-catalog](http://www.sdss.org/dr16/data_access/value-added-catalogs/?vac_id=apogee-red-clump-rc-catalog)

Data Model (DR16): [https://data.sdss.org/datamodel/files/APOGEE\\_RC/cat/apogee-rc-DR16.html](https://data.sdss.org/datamodel/files/APOGEE_RC/cat/apogee-rc-DR16.html)

`astroNN.datasets.apogee.load_apogee_rc(dr=None, unit='distance', extinction=True)`

Load apogee red clumps (absolute magnitude measurement)

### Parameters

- **dr** (*int*) – Apogee DR
- **unit** (*string*) – which unit 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 unit is NOT 'distance'

### Returns

numpy array of ra, dec, array

### Return type

ndarrays

### History

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

2018-May-12 - Updated - Henry Leung (University of Toronto)

```
1 from astroNN.apogee import apogee_rc
2
3 local_path_to_file = apogee_rc(dr=16)
```

Or you can use `load_apogee_rc()` to load the data by

```
1 from astroNN.datasets import load_apogee_rc
2
3 # unit can be 'distance' for distance in parsec, 'absmag' for k-band absolute magnitude
4 # 'fakemag' for astroNN's k-band fakemag scale
5 RA, DEC, array = load_apogee_rc(dr=16, unit='distance', extinction=True) # extinction_
  ↳ only effective if not unit='distance'
```

## APOKASC in the Kepler Fields

```
1 from astroNN.datasets import load_apokasc
2
3 ra, dec, logg = load_apokasc()
4
5 # OR you want the gold and basic standard separately
6 gold_ra, gold_dec, gold_logg, basic_ra, basic_dec, basic_logg = load_
  ↳ apokasc(combine=False)
```

## APOGEE Distance Estimations

Introduction: [http://www.sdss.org/dr14/data\\_access/value-added-catalogs/?vac\\_id=apogee-dr14-based-distance-estimations](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](https://data.sdss.org/datamodel/files/APOGEE_DISTANCES/apogee_distances.html) Data Model (DR16): [https://data.sdss.org/datamodel/files/APOGEE\\_STARHORSE/apogee\\_starhorse.html](https://data.sdss.org/datamodel/files/APOGEE_STARHORSE/apogee_starhorse.html)

`astroNN.apogee.apogee_distances(dr=None, flag=None)`

Download the APOGEE Distances VAC catalogue (APOGEE Distances for DR14, APOGEE Starhorse for DR16/17)

### 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)

2021-Jan-29 - Updated - Henry Leung (University of Toronto)

```
1 from astroNN.apogee.downloader import apogee_distances
2
3 local_path_to_file = apogee_distances(dr=14)
```

`astroNN.datasets.load_apogee_distances(dr=None, unit='distance', cuts=True, extinction=True, keepdims=False)`

Load apogee distances (absolute magnitude from stellar model)

### Parameters

- **dr** (*int*) – Apogee DR
- **unit** (*string*) – which unit 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 unit 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, array, err\_array

**Return type**  
ndarrays

**History**

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

2021-Jan-29 - Updated - Henry Leung (University of Toronto)

Or you can use `load_apogee_distances()` to load the data by

```
1 from astroNN.datasets import load_apogee_distances
2
3 # unit can be 'distance' for distance in parsec, 'absmag' for k-band absolute magnitude
4 # 'fakemag' for astroNN's k-band fakemag scale
5 # cuts=True to cut out those unknown values (-9999.) and measurement error > 20%
6 RA, DEC, array, err_array = load_apogee_distances(dr=14, unit='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](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](https://data.sdss.org/datamodel/files/APOGEE_REDUX/APRED_VERS/APSTAR_VERS/ASPCAP_VERS/RESULTS_VERS/CANNON_VERS/allStarCannon.html)

`astroNN.apogee.allstar_cannon(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)

```
1 from astroNN.apogee import allstar_cannon
2
3 local_path_to_file = allstar_cannon(dr=14)
```

## 1.15 Mini Tools for LAMOST data

`astroNN.lamost` 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](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>

### 1.15.1 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)
```

### 1.15.2 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 uncertainty

#### 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)
```

### 1.15.3 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
```

## 1.16 Mini Tools for Gaia data

---

**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](#)

---

`astroNN.gaia` module provides a handful of tools to deal with astrometry and photometry. 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/>

### 1.16.1 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)

```

1 from astroNN.gaia import gaiadr2_parallax
2
3 # To load Gaia DR2 - APOGEE DR14 matches, indices corresponds to APOGEE allstar DR14
  ↳ file
4 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

```

1 from astroNN.gaia import tgas
2
3 # To download tgas dr1 to GAIA_TOOLS_DATA and it will return the list of path to those
```

(continues on next page)

(continued from previous page)

```
↪ files
4 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)

```
1 from astroNN.gaia import tgas_load
2
3 # To load the tgas DR1 files and return a dictionary of ra(J2015), dec(J2015), pmra, ↪
↪ pmdec, parallax, parallax error, g-band mag
4 # cuts=True to cut bad data (negative parallax and percentage error more than 20%)
5 output = tgas_load(cuts=True)
6
7 # outout dictionary
8 output['ra'] # ra(J2015)
9 output['dec'] # dec(J2015)
10 output['pmra'] # proper motion in RA
11 output['pmdec'] # proper motion in DEC
12 output['parallax'] # parallax
13 output['parallax_err'] # parallax error
14 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](https://github.com/jobovy/gaia_tools)

```
1 from astroNN.gaia import gaia_source
2
3 # To download gaia_source DR1 to GAIA_TOOLS_DATA and it will return the list of path to ↪
↪ those files
4 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

```

1 from astroNN.gaia import anderson_2017_parallax
2
3 # To load the improved parallax
4 # Both parallax and para_var is in mas
5 # cuts=True to cut bad data (negative parallax and percentage error more than 20%)
6 ra, dec, parallax, para_err = anderson_2017_parallax(cuts=True)

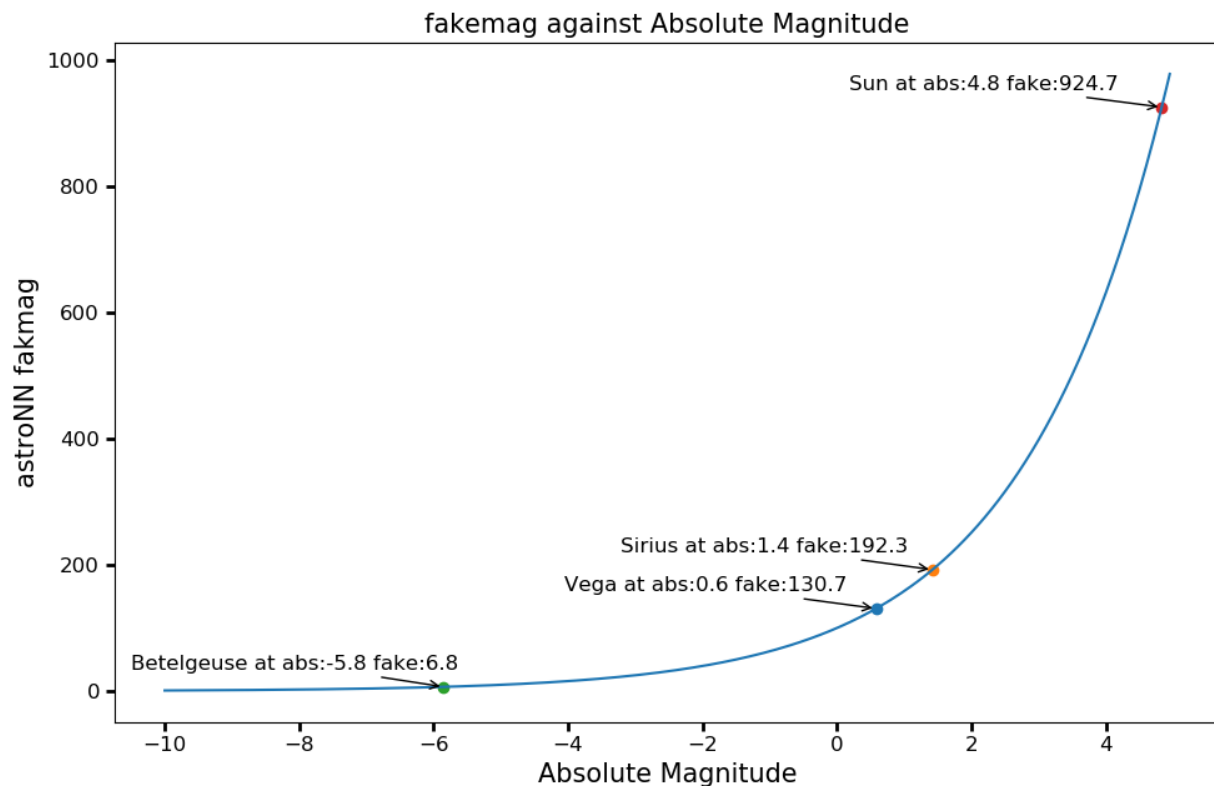
```

### 1.16.2 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  $\varpi$  is parallax in *mas*

You can get a sense of the fakemag scale from the following plot



### 1.16.3 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]*) – astroNN 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 log10 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 log10 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

```
1 from astroNN.gaia import ...
```

Preserving `magicnumber` means the indices which matched `magicnumber` in `config.ini` will be preserved, for example:

```
1 from astroNN.gaia import absmag_to_pc
2
3 print(absmag_to_pc([1., -9999.], [2., 1.]))
4 >>> <Quantity [15.84893192, -9999.] pc>
5
6 print(absmag_to_pc([1., -9999.], [-9999., 1.]))
7 >>> <Quantity [-9999., -9999.] pc>
```

Since some functions support astropy Quantity framework, you can convert between units easily. Example:

```
1 from astroNN.gaia import absmag_to_pc
2 from astropy import units as u
3 import numpy as np
4
5 # Example data of [Vega, Sirius, Betelgeuse]
6 absmag = np.array([0.582, 1.42, -5.85])
7 mag = np.array([0.03, -1.46, 0.5])
8 pc = absmag_to_pc(absmag, mag) # The output - pc - carries astropy unit
9
10 # Convert to AU
11 distance_in_AU = pc.to(u.AU)
12
13 # Or convert to angle units by using astropy's equivalencies function
14 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

```
1 from astroNN.gaia import fakemag_to_pc
2
3 fakemag = 300
```

(continues on next page)

(continued from previous page)

```

4 fakemag_err = 100
5 apparent_mag = 10
6
7 print(fakemag_to_pc(fakemag, apparent_mag, fakemag_err))
8 >>> (<Quantity 333.33333333 pc>, <Quantity 111.11111111 pc>)

```

### 1.16.4 Coordinates Matching between catalogs xmatch

`astroNN.datasets.xmatch.xmatch(ra1, dec1, ra2, dec2, epoch1=2000.0, epoch2=2000.0, pmra2=None, pmdec2=None, maxdist=2)`

Cross-matching between arrays by RA/DEC coordinates

#### Parameters

- **ra1** (*ndarray*) – 1d array for the first catalog RA
- **dec1** (*ndarray*) – 1d array for the first catalog DEC
- **ra2** (*ndarray*) – 1d array for the second catalog RA
- **dec2** (*ndarray*) – 1d array for the second catalog DEC
- **epoch1** (*Union([float, ndarray])*) – Epoch for the first catalog, can be float or 1d array
- **epoch2** – Epoch for the second catalog, can be float or 1d array
- **pmra2** (*ndarray*) – RA proper motion for second catalog, only effective if *epoch1* not equals *epoch2*
- **pmdec2** (*ndarray*) – DEC proper motion for second catalog, only effective if *epoch1* not equals *epoch2*
- **maxdist** (*float*) – Maximum distance in arcsecond

#### Returns

numpy array of ra, dec, separation

#### Return type

ndarrays

#### History

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

2021-Jan-29 - Updated - Henry Leung (University of Toronto)

Here is an example

```

1 from astroNN.datasets import xmatch
2 import numpy as np
3
4 # Some coordinates for cat1, J2000.
5 cat1_ra = np.array([36., 68., 105., 23., 96., 96.])
6 cat1_dec = np.array([72., 56., 54., 55., 88., 88.])
7
8 # Some coordinates for cat2, J2000.
9 cat2_ra = np.array([23., 56., 222., 96., 245., 68.])
10 cat2_dec = np.array([36., 68., 82., 88., 26., 56.])

```

(continues on next page)

(continued from previous page)

```

11
12 # Using maxdist=2 arcsecond separation threshold, because its default, so not shown here
13 # Using epoch1=2000. and epoch2=2000., because its default, so not shown here
14 # because both datasets are J2000., so no need to provide pmra and pmdec which
  ↳ represent proper motion
15 idx_1, idx_2, sep = xmatch(ra1=cat1_ra, dec1=cat1_dec, ra2=cat2_ra, dec2=cat2_dec)
16
17 print(idx_1)
18 >>> [1 4 5]
19 print(idx_2)
20 >>> [5 3 3]
21 print(cat1_ra[idx_1], cat2_ra[idx_2])
22 >>> [68. 96. 96.], [68. 96. 96.]
23
24 # What happens if we swap cat_1 and cat_2
25 idx_1, idx_2, sep = xmatch(ra1=cat2_ra, dec1=cat2_dec, ra2=cat1_ra, dec2=cat1_dec)
26
27 print(idx_1)
28 >>> [3 5]
29 print(idx_2)
30 >>> [4 1]
31 print(cat1_ra[idx_2], cat2_ra[idx_1])
32 >>> [96. 68.], [96. 68.] # xmatch cant find all the match

```

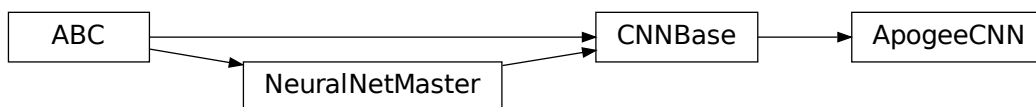
## 1.17 APOGEE Spectra with Convolutional Neural Net - ApogeeCNN

**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

```

(continues on next page)



(continued from previous page)

```

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
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 calculate jacobian which represents the output derivative to the input and see where those output is sensitive to in inputs.

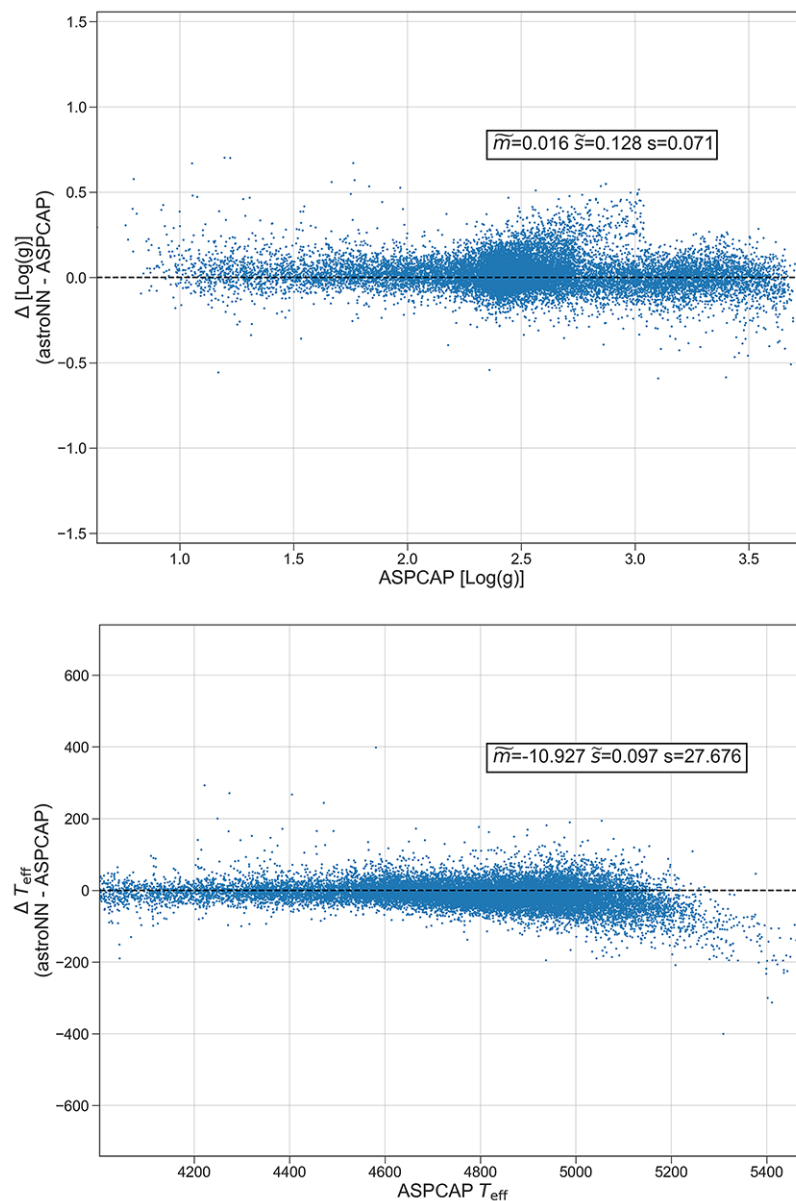
```
# Calculate jacobian first
jacobian_array = cnn_net.jacobian(x_test, mean_output=True)
```

---

**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()`)

---

### 1.17.1 Example Plots using `aspcap_residue_plot`



### 1.17.2 ASPCAP labels prediction using CNN vs The Cannon 2

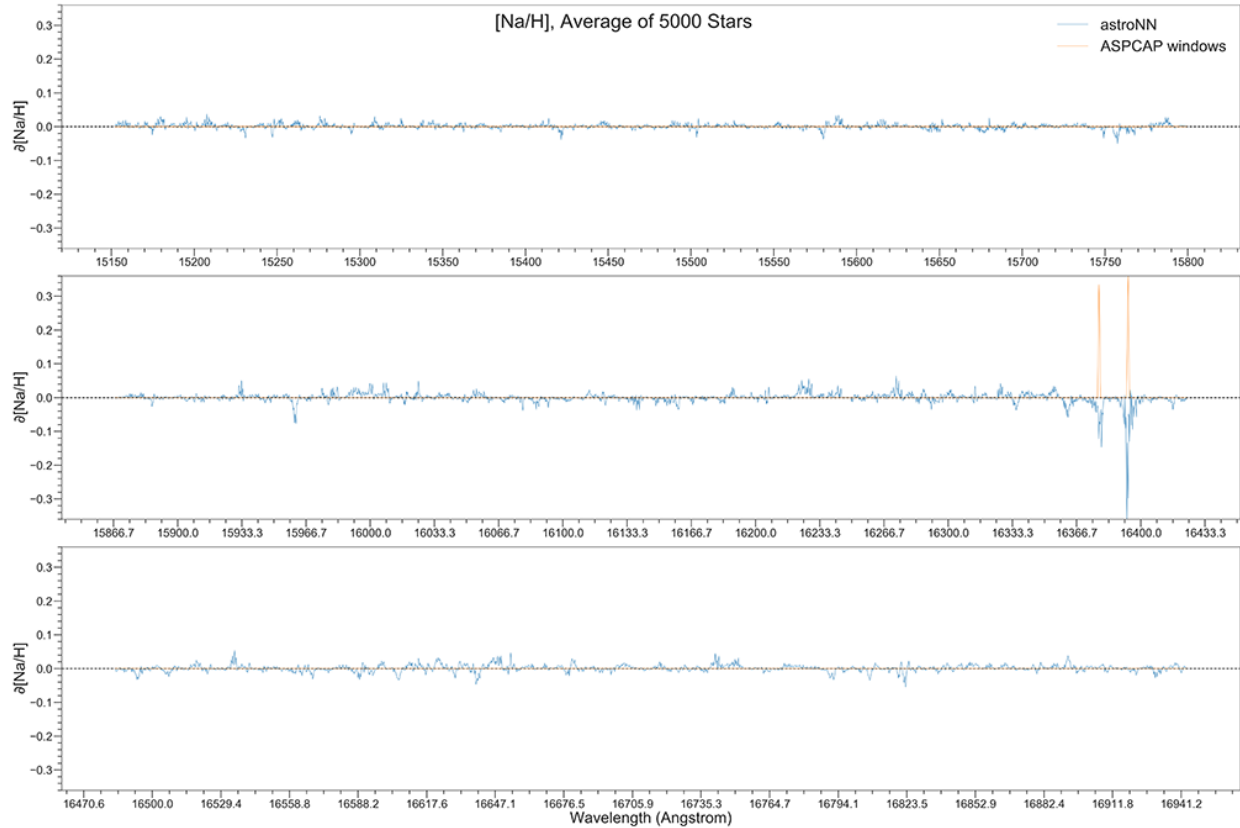
**Warning:** Please refer to Bayesian Neural Network for the most updated result: [http://astronn.readthedocs.io/en/latest/neuralnets/apogee\\_bcnn.html](http://astronn.readthedocs.io/en/latest/neuralnets/apogee_bcnn.html)

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

### 1.17.3 Example Plots using jacobian





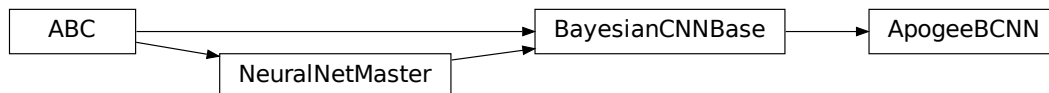
## 1.18 APOGEE Spectra with Bayesian Neural Net - ApogeeBCNN

`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

```

```

# Load the train data from dataset first, x_train is spectra and y_train will be ASPCAP_

```

(continues on next page)

(continued from previous page)

```

↪ 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')

```

(continues on next page)

(continued from previous page)

```

loader2.load_combined = False
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.

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)

```

**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()`)

### 1.18.1 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`,  $4000 < \text{Teff} < 5500$ ,  $200 < \text{SNR}$

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

### 1.18.2 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

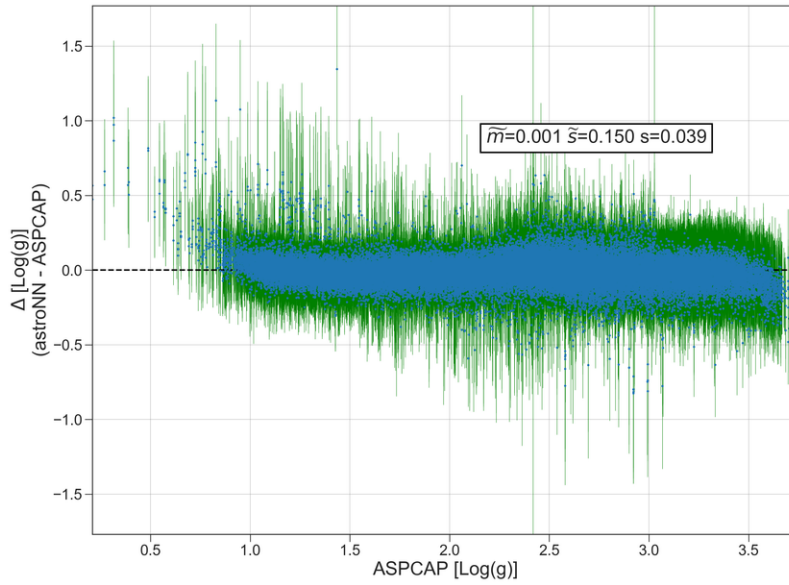
### 1.18.3 ASPCAP Labels Prediction with limited amount of data

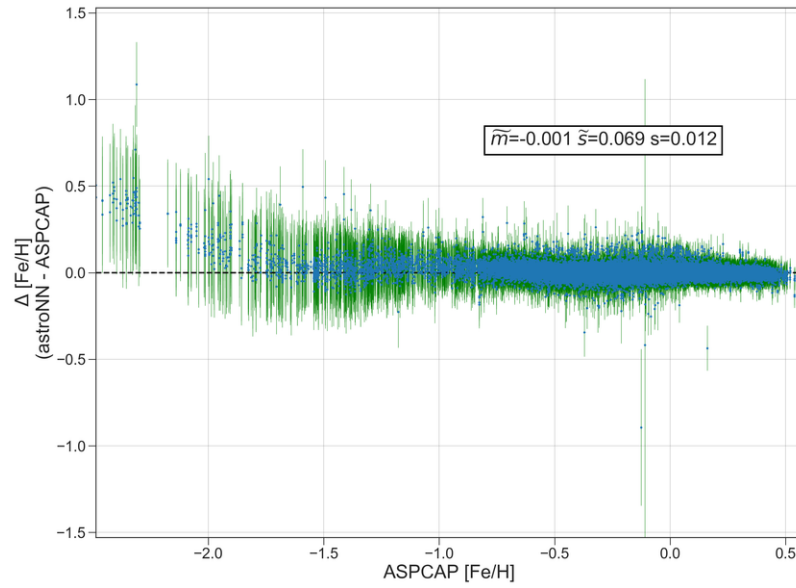
Internal model identifier for the author: astroNN\_0401\_run001

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

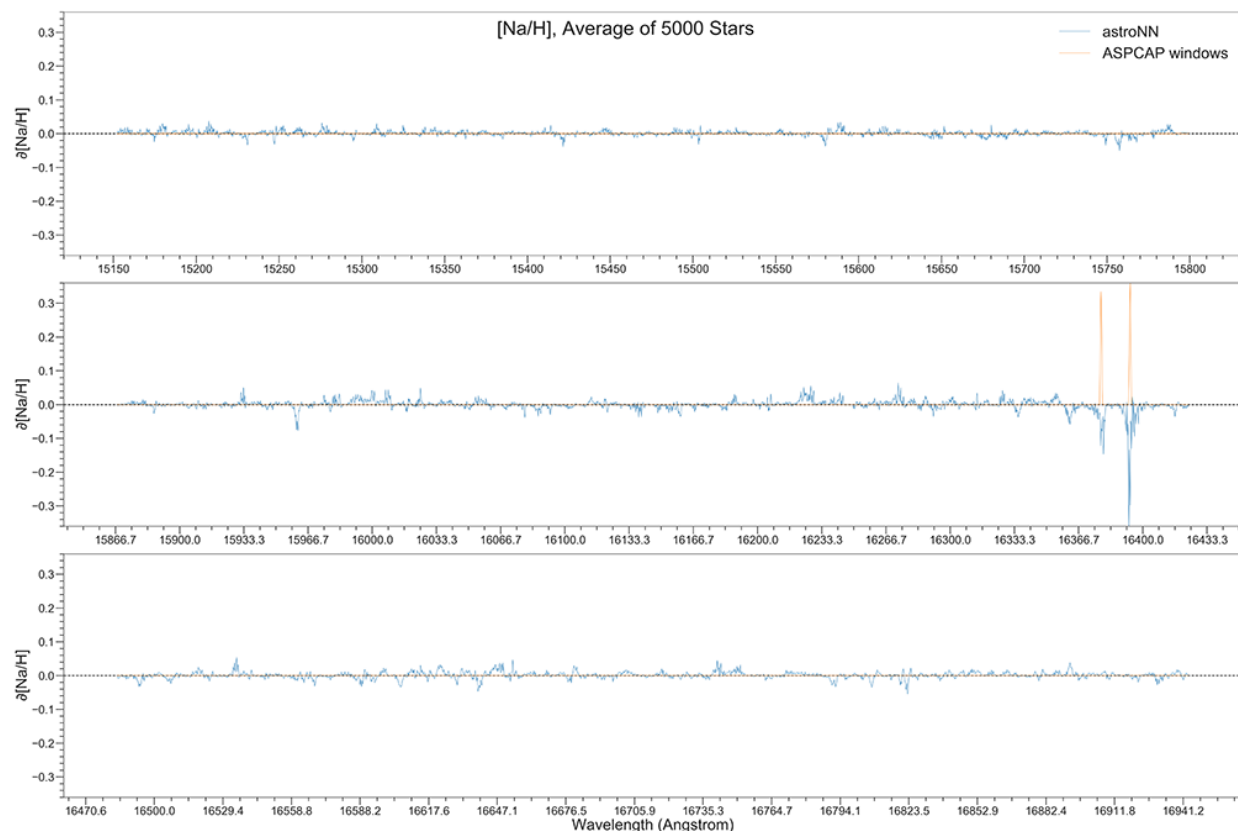
#### 1.18.4 Example Plots using aspcap\_residue\_plot





### 1.18.5 Example Plots using jacobian





## 1.19 APOGEE Spectra with Censored Bayesian NN - ApogeeBCNNCensored

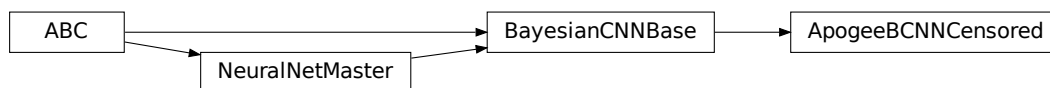
**class** astroNN.models.apogee\_models.**ApogeeBCNNCensored**(*lr=0.0005, dropout\_rate=0.3*)

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/2019MNRAS.483.3255L/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
```

(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 = 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')
```

(continues on next page)

(continued from previous page)

```

# 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
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 = bcnnccensored_net.test(x_test)

```

```
bcnnccensored_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.

```

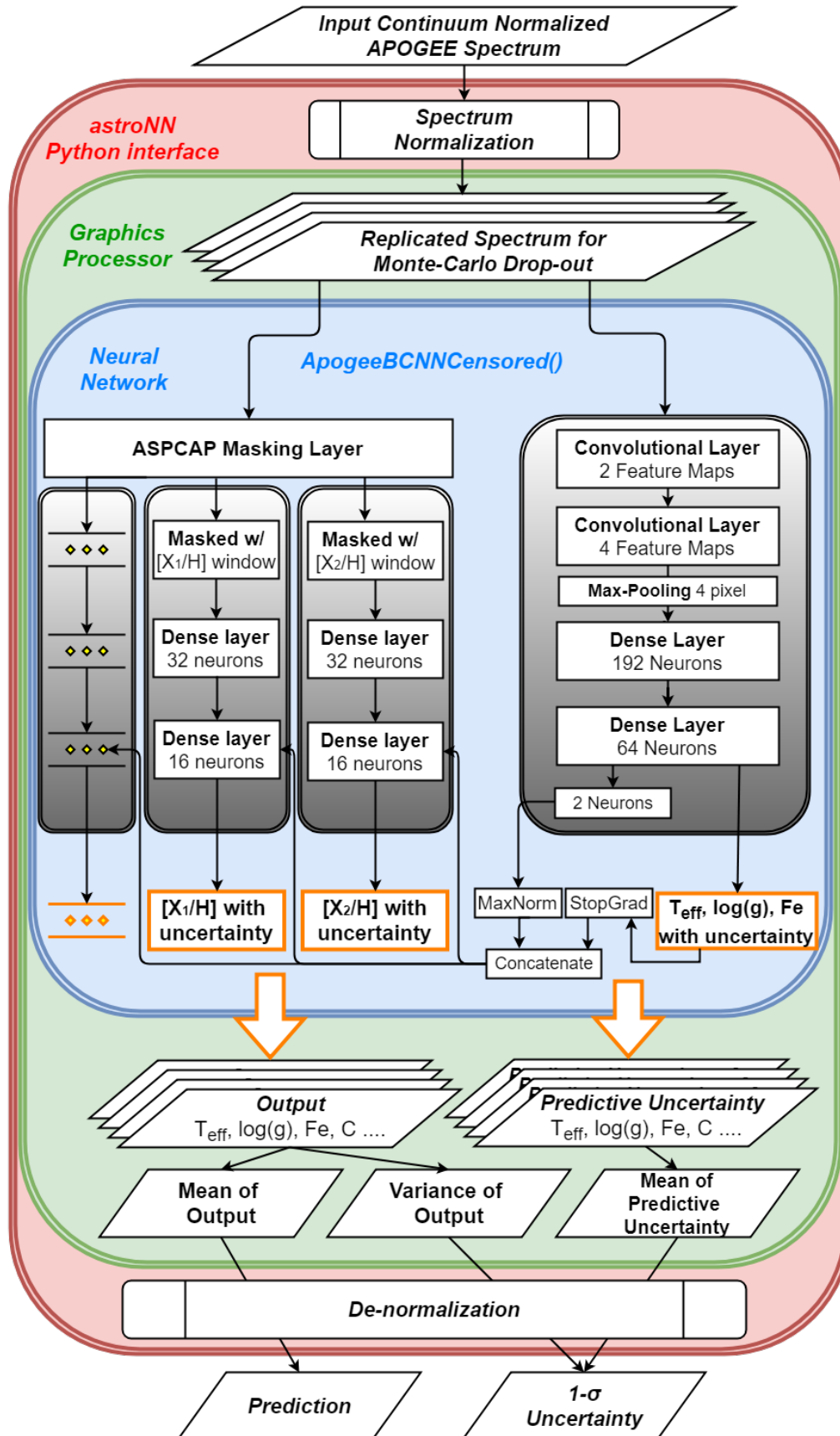
# Calculate jacobian first
jacobian_array = bcnnccensored_net.jacobian(x_test, mean_output=True)

# Plot the graphs
bcnnccensored_net.jacobian_aspcap(jacobian=jacobian_array, dr=14)

```

### 1.19.1 Architecture

The architecture of this neural network is as follow.

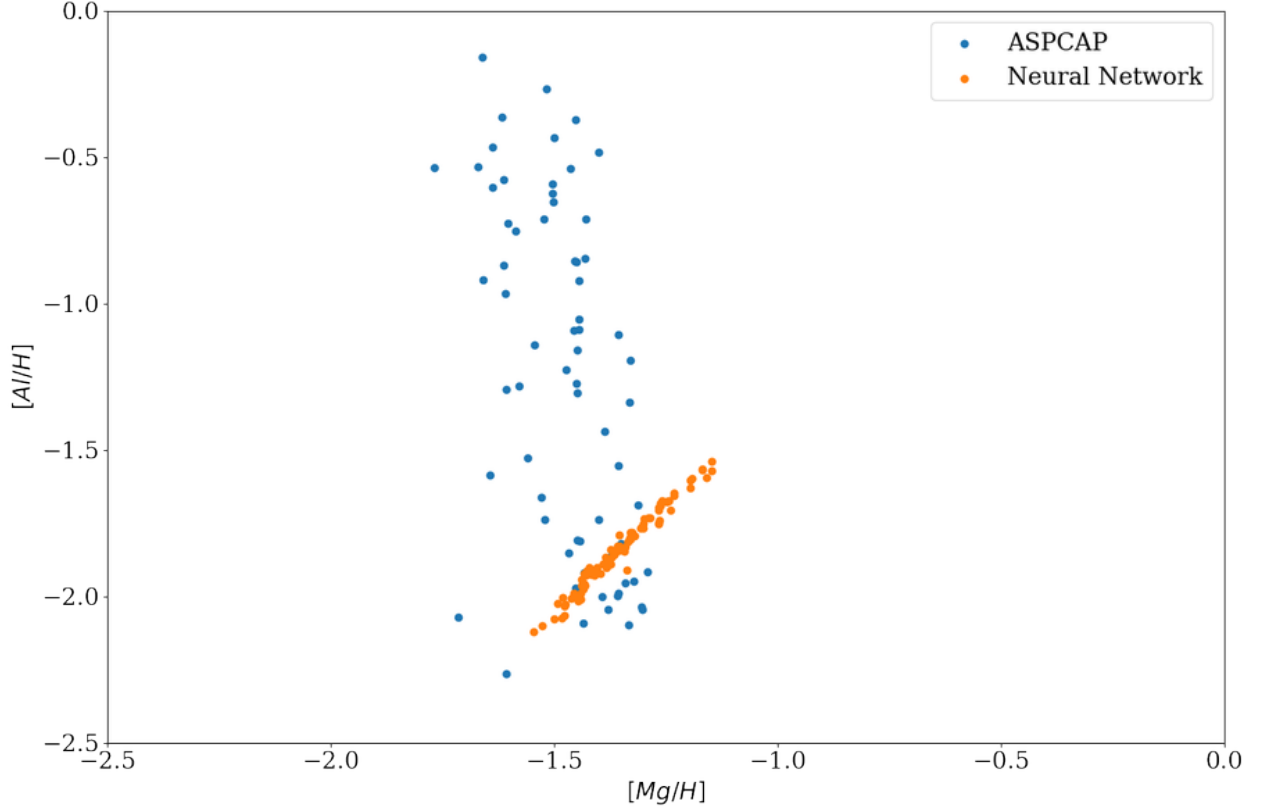




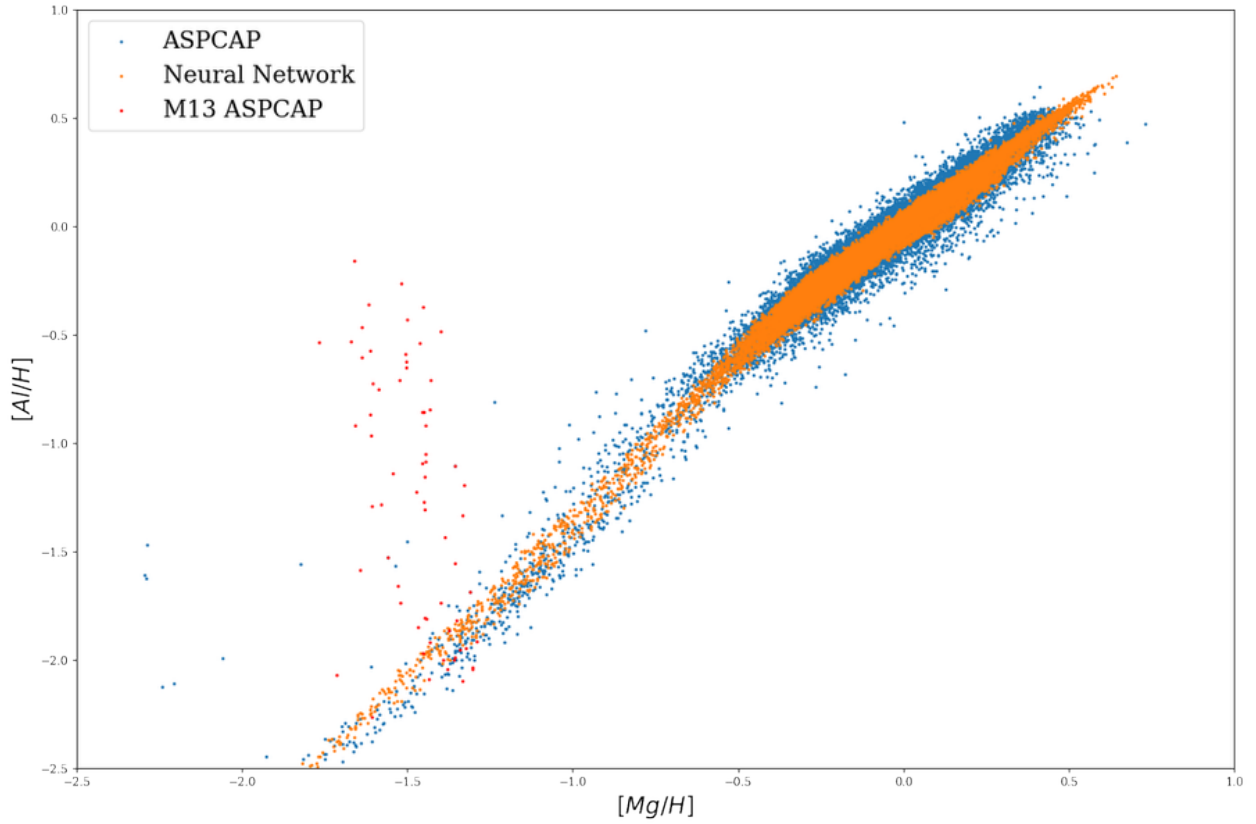
### 1.19.2 Why Censored NN for APOGEE Spectra analysis?

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.

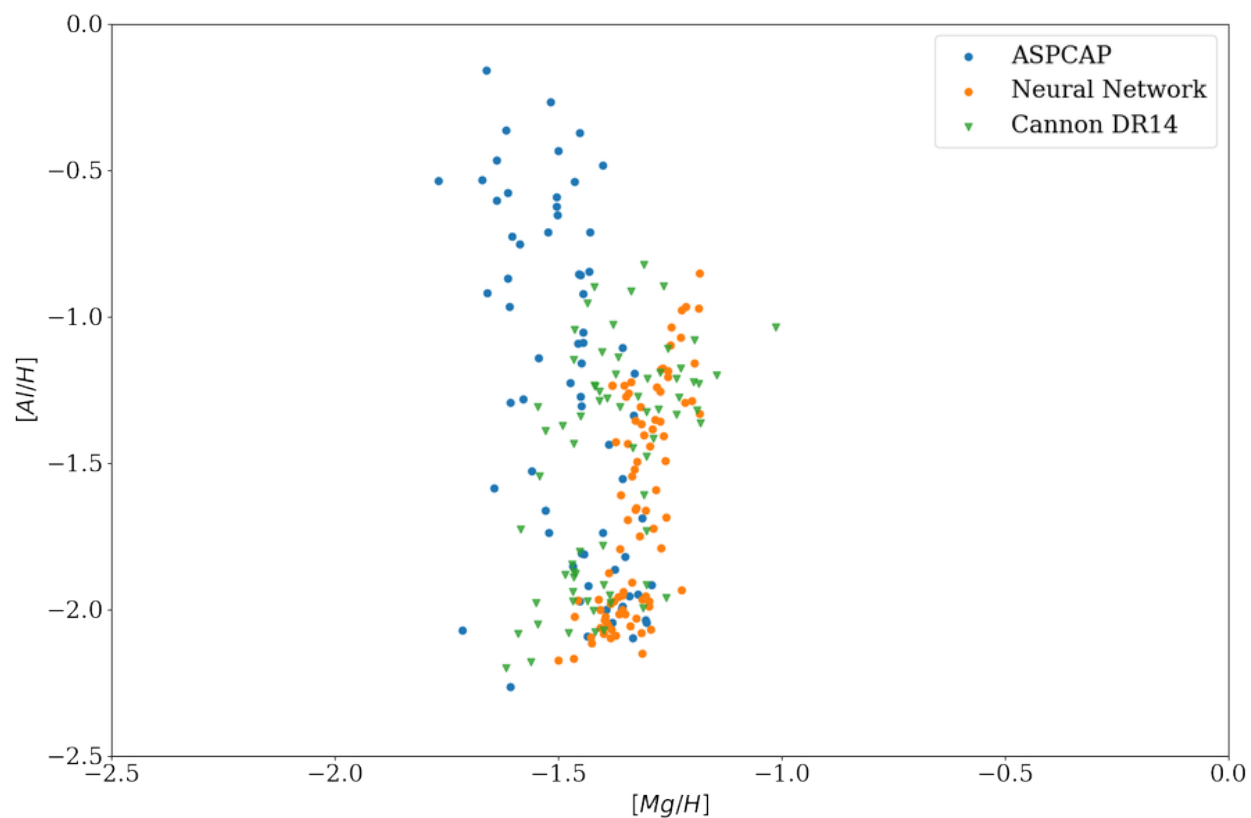


It becomes clear when we plot the training set  $[Al/H]$  vs  $[Mg/H]$  as follow,  $[Al/H]$  and  $[Mg/H]$  are strongly correlated and *ApogeeBCNN* is just measuring  $[Al/H]$  as some kind of  $[Mg/H]$  and fooled in *M13* because *M13* has a spread in  $[Al/H]$  but not  $[Mg/H]$ , in other word, the region in  $[Mg/H, Al/H]$  parameter space of *M13* is not covered by training set.

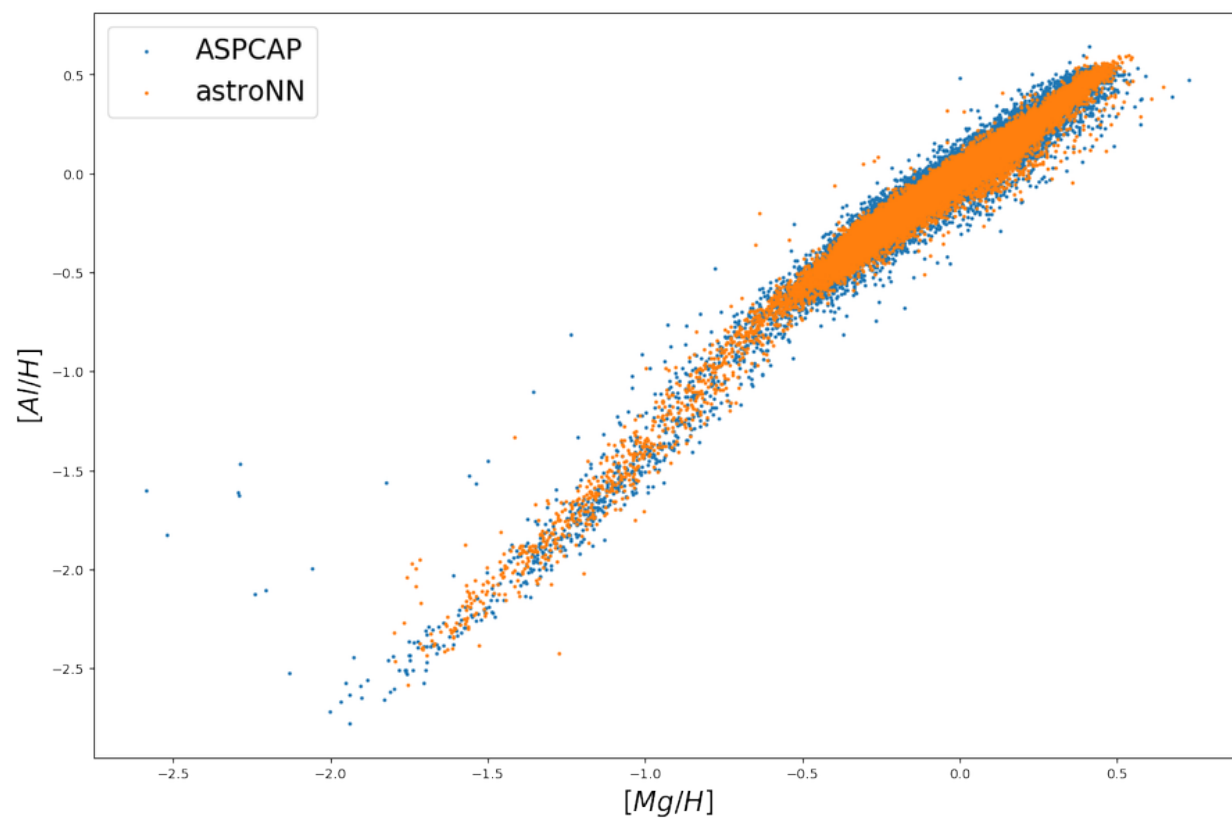


So Censored Neural Net is proposed to solve the issue by encouraging neural network to look at the ASPCAP window regions.

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



### 1.19.3 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 - 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

## 1.20 APOGEE Spectra with Bayesian NN and Gaia offset calibration - 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')

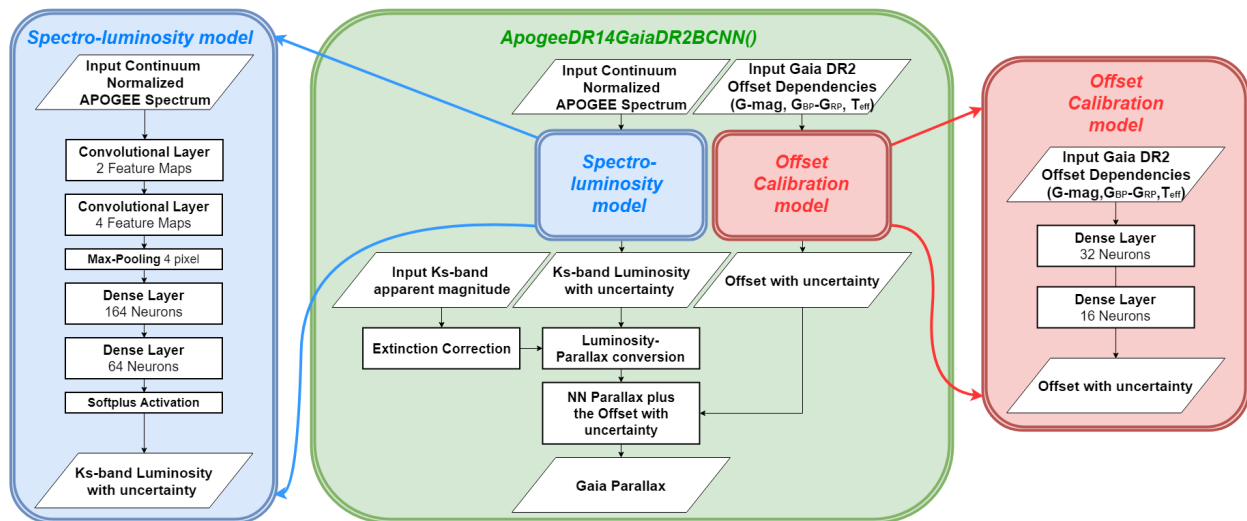
# Load the test data from dataset, x_test is spectra and y_test will be ASPCAP labels
test_data = .....

# 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)
```

### 1.20.1 Architecture

The architecture of this neural network is as follow.



## 1.21 Convolutional Variational Autoencoder - ApogeeCVAE

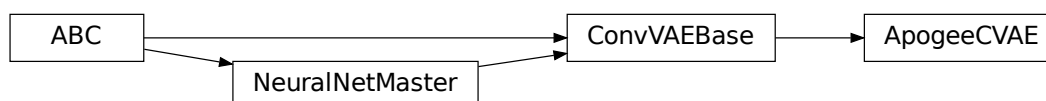
**Warning:** Information are obsolete, the following code may not be able to run properly with astroNN latest commit

**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()
  
```

### 1.21.1 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)
```

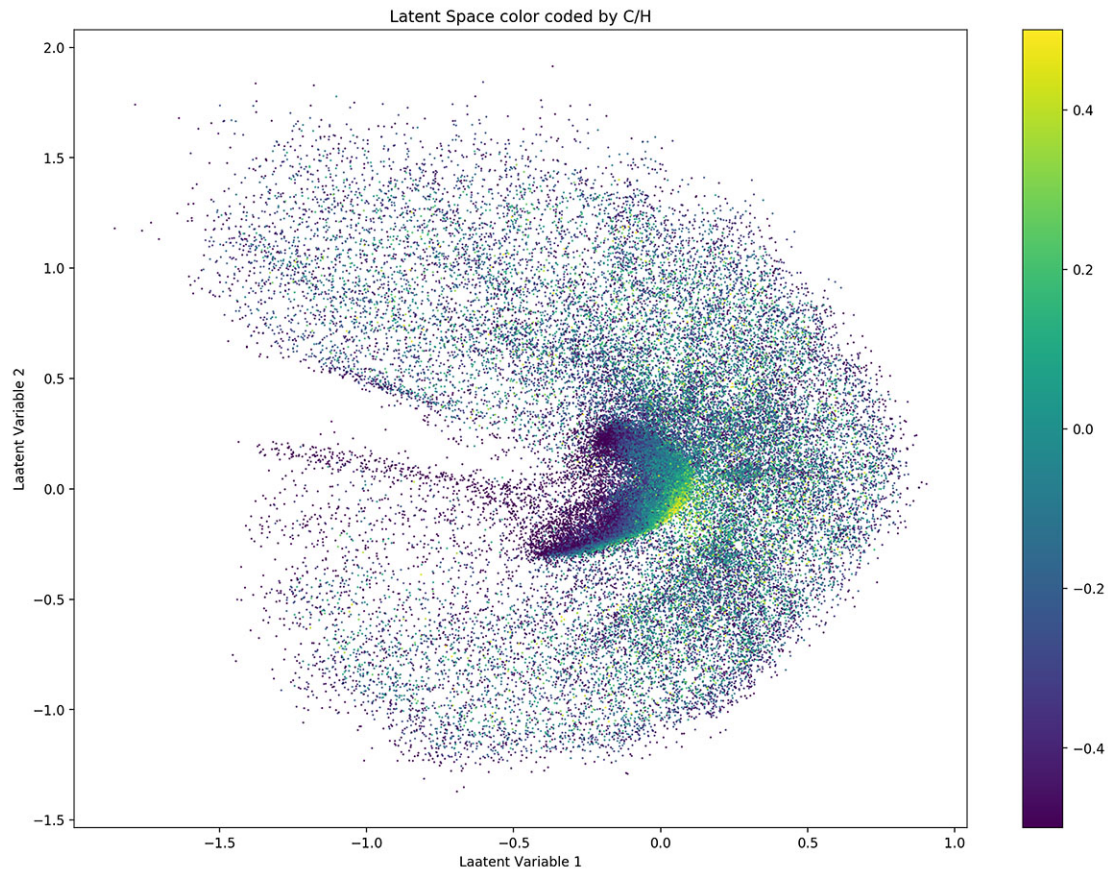
---

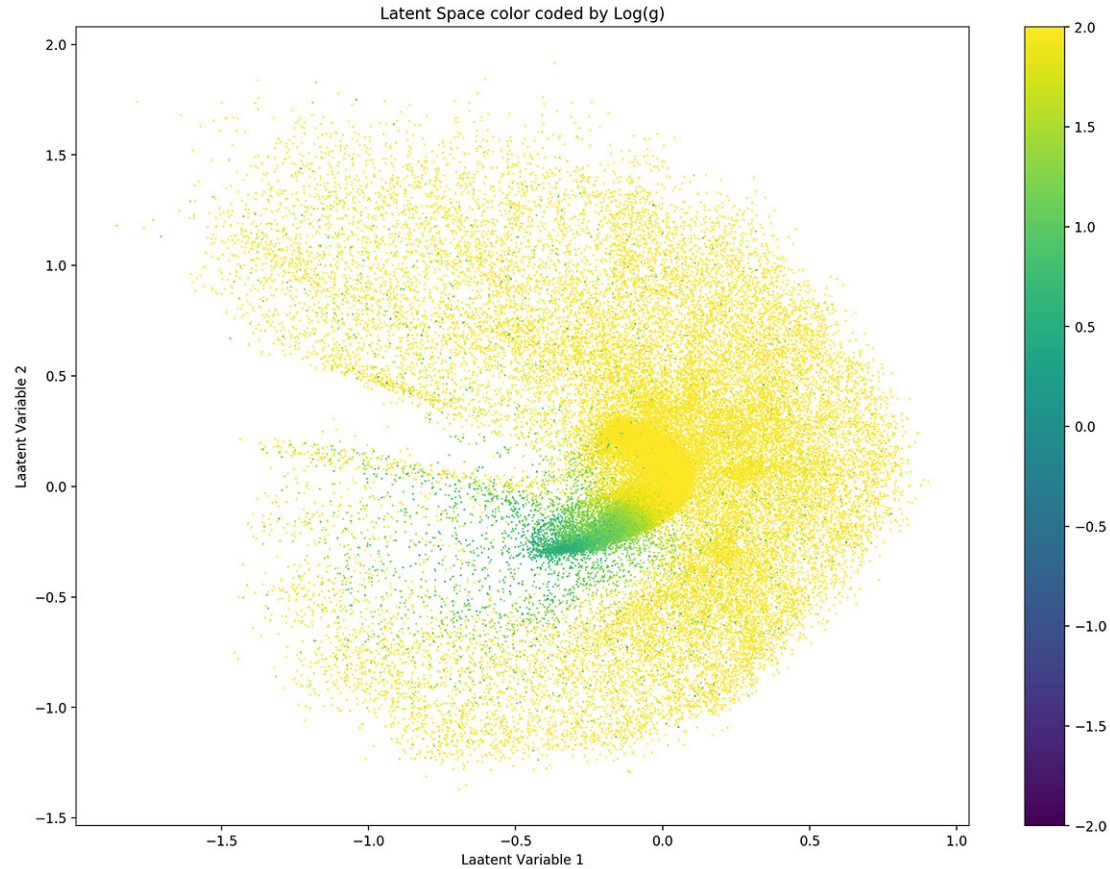
**Note:** You can access to Keras model method like model.predict via (in the above tutorial) vae\_net.keras\_model (Example: vae\_net.keras\_model.predict())

---



### 1.21.2 Example Plots on latent space using `VAE.plot_latent()`



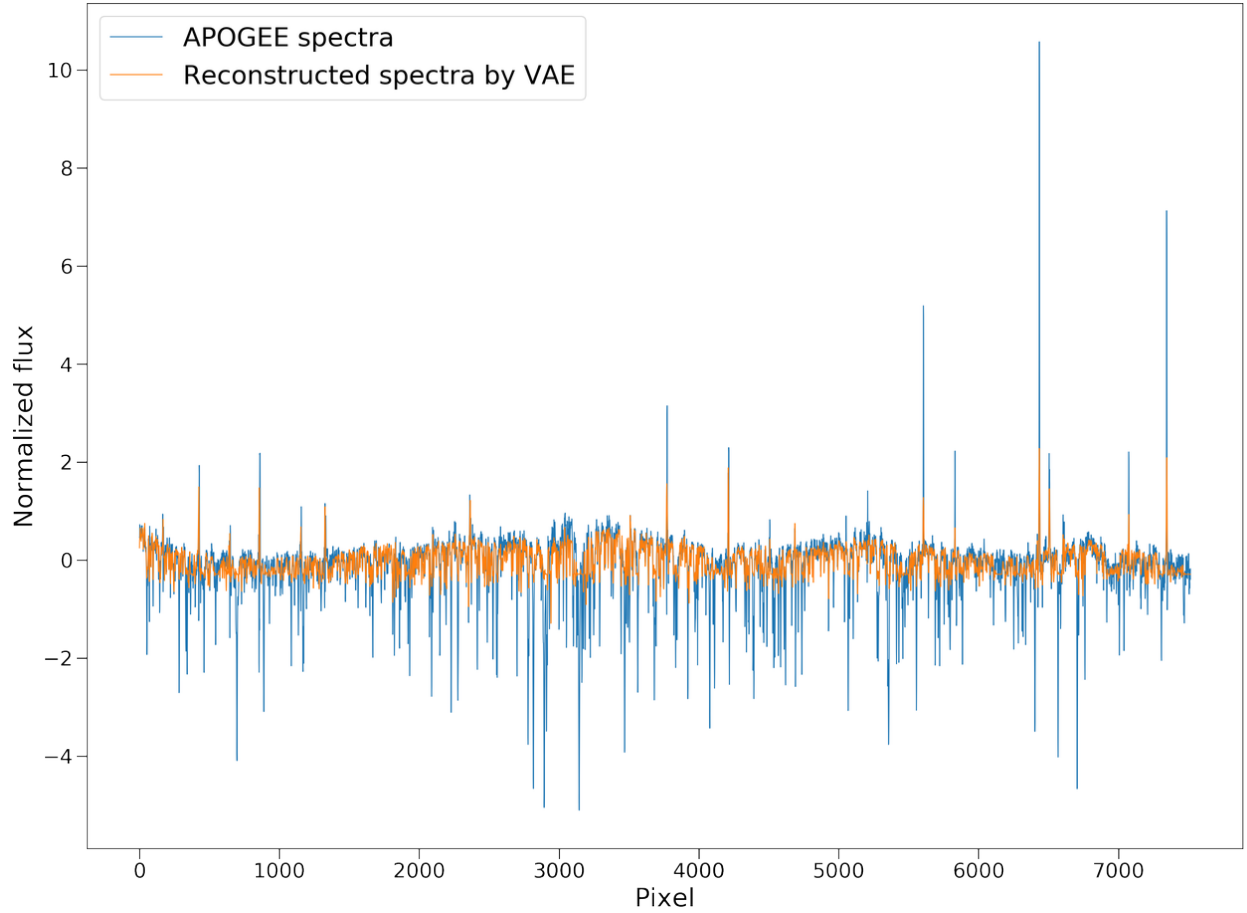


### 1.21.3 Example Plots on spectra reconstruction

```
x_re = cvae_net.test(x_test)

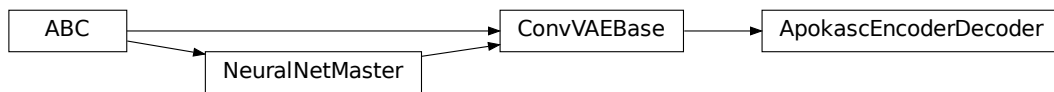
import pylab as plt

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)
```



## 1.22 Encoder-decoder for APOGEE and Kepler - ApokascEncoderDecoder

`class astroNN.models.apogee_models.ApokascEncoderDecoder(lr=0.0005, dropout_rate=0.0)`



*ApokascEncoderDecoder* can only be used with Apogee spectra with 7,514 pixels and Kepler PSD with 2,092. Both numbers are **hardcoded** into the model

Please refers to the paper <https://ui.adsabs.harvard.edu/abs/2023arXiv230205479L/abstract> and [https://github.com/henrysky/astroNN\\_ages](https://github.com/henrysky/astroNN_ages) for detail

```
from astroNN.models import ApokascEncoderDecoder
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 = True
x_train, y_train, x_err, y_err = loader.load()

# And then create an instance of Bayesian Convolutional Neural Network class
ved = ApokascEncoderDecoder()

# You don't have to specify the task because its 'regression' by default. But if you are_
↳ doing classification. you can set task='classification'
ved.task = 'regression'

# Set max_epochs to 10 for a quick result. You should train more epochs normally,
↳ especially with dropout
ved.max_epochs = 10
ved.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

```
ved.batch_size = 128
ved.initializer = 'glorot_uniform'
ved.activation = 'relu'
ved.num_filters = [32, 64, 16, 16]
ved.filter_len = [8, 32]
ved.pool_length = 2
ved.num_hidden = [16, 16]
ved.latent_dim = 5
ved.max_epochs = 100
ved.lr = 0.005
ved.reduce_lr_epsilon = 0.00005
ved.reduce_lr_min = 0.0000000001
ved.reduce_lr_patience = 10
ved.target = 'PSD'
ved.l2 = 5e-9
ved.input_norm_mode = 2
ved.labels_norm_mode = 0
```

---

**Note:** You can disable astroNN data normalization via `ApokascEncoderDecoder.input_norm_mode=0` as well as `ApokascEncoderDecoder.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 `ved` 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
ved = load_folder('astroNN_0101_run001')
```

(continues on next page)

(continued from previous page)

```
# Load the test data from dataset, x_test is APOGEE spectra
# something here

# pred contains denormalized result aka. Kepler PSD prediction in this case
pred = ved.test(x_test)

# methods like predict_encoder() and predict_decoder() also available
```

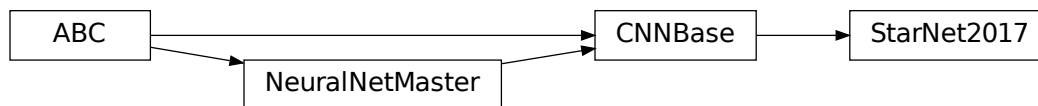
## 1.23 StarNet (arXiv:1709.09182)

**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

---

## 1.24 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)
```

## ACKNOWLEDGING ASTRONN

Please cite the following paper that describes astroNN if astroNN is used in your research as well as linking it to <https://github.com/henrysky/astroNN>

**Deep learning of multi-element abundances from high-resolution spectroscopic data** [arXiv:1808.04428][ADS]

And here is a list of publications using astroNN - *Publications using astroNN*





## AUTHORS

- **Henry Leung** - *Initial work and developer* - [henrysky](#)  
Department of Astronomy & Astrophysics, University of Toronto  
Contact Henry: [henrysky.leung \[at\] utoronto.ca](mailto:henrysky.leung@utoronto.ca)
- **Jo Bovy** - *Project Supervisor* - [jobovy](#)  
Department of Astronomy & Astrophysics, University of Toronto



## PYTHON MODULE INDEX

### a

- `astroNN.apogee`, 96
- `astroNN.apogee.chips`, 96
- `astroNN.apogee.downloader`, 102
- `astroNN.config`, 15
- `astroNN.datasets.galaxy10`, 3
- `astroNN.datasets.galaxy10sdss`, 7
- `astroNN.gaia`, 110
- `astroNN.gaia.downloader`, 110
- `astroNN.lamost`, 108
- `astroNN.models`, 67
- `astroNN.models.apogee_models`, 147
- `astroNN.neuralode`, 65
- `astroNN.neuralode.odeint`, 65
- `astroNN.nn`, 60
- `astroNN.nn.callbacks`, 57
- `astroNN.nn.layers`, 38
- `astroNN.nn.losses`, 22
- `astroNN.nn.metrics`, 22
- `astroNN.nn.numpy`, 61
- `astroNN.nn.utilities`, 57
- `astroNN.nn.utilities.normalizer`, 58



## Symbols

`__call__()` (*astroNN.nn.layers.FastMCInference*  
method), 52

## A

`absmag_to_fakemag()` (*in module astroNN.gaia*), 115  
`absmag_to_logsol()` (*in module astroNN.gaia*), 116  
`absmag_to_pc()` (*in module astroNN.gaia*), 114  
`allstar()` (*in module astroNN.apogee*), 103  
`allstar_cannon()` (*in module astroNN.apogee*), 108  
`allvisit()` (*in module astroNN.apogee*), 103  
`apogee_astroNN()` (*in module astroNN.apogee.downloader*), 105  
`apogee_continuum()` (*in module astroNN.apogee*), 96  
`apogee_distances()` (*in module astroNN.apogee*), 107  
`ApogeeBCNN` (*class in astroNN.models.apogee\_models*), 125  
`ApogeeBCNNCensored` (*class in astroNN.models.apogee\_models*), 133  
`ApogeeCNN` (*class in astroNN.models.apogee\_models*), 120  
`ApogeeCVAE` (*class in astroNN.models.apogee\_models*), 143  
`ApogeeDR14GaiaDR2BCNN` (*class in astroNN.models.apogee\_models*), 140  
`ApokascEncoderDecoder` (*class in astroNN.models.apogee\_models*), 147  
`aspcap_mask()` (*in module astroNN.apogee*), 102  
`astroNN.apogee`  
    module, 96  
`astroNN.apogee.chips`  
    module, 96  
`astroNN.apogee.downloader`  
    module, 102  
`astroNN.config`  
    module, 15  
`astroNN.datasets.galaxy10`  
    module, 3  
`astroNN.datasets.galaxy10sdss`  
    module, 7  
`astroNN.gaia`  
    module, 110

`astroNN.gaia.downloader`  
    module, 110  
`astroNN.lamost`  
    module, 108  
`astroNN.models`  
    module, 67  
`astroNN.models.apogee_models`  
    module, 120, 125, 133, 140, 142, 147, 149  
`astroNN.neuralode`  
    module, 65  
`astroNN.neuralode.odeint`  
    module, 65  
`astroNN.nn`  
    module, 57, 60  
`astroNN.nn.callbacks`  
    module, 57  
`astroNN.nn.layers`  
    module, 38  
`astroNN.nn.losses`  
    module, 22  
`astroNN.nn.metrics`  
    module, 22  
`astroNN.nn.numpy`  
    module, 61  
`astroNN.nn.utilities`  
    module, 57  
`astroNN.nn.utilities.normalizer`  
    module, 58

## B

`bayesian_binary_crossentropy_var_wrapper()`  
    (*in module astroNN.nn.losses*), 35  
`bayesian_binary_crossentropy_wrapper()` (*in*  
    *module astroNN.nn.losses*), 34  
`bayesian_categorical_crossentropy_var_wrapper()`  
    (*in module astroNN.nn.losses*), 33  
`bayesian_categorical_crossentropy_wrapper()`  
    (*in module astroNN.nn.losses*), 32  
`BayesianCNNBase` (*class in astroNN.models.base\_bayesian\_cnn*), 74  
`binary_accuracy()` (*in module astroNN.nn.losses*), 37

`binary_crossentropy()` (in module `astroNN.nn.losses`), 31  
`bitmask_boolean()` (in module `astroNN.apogee`), 100  
`bitmask_decompositor()` (in module `astroNN.apogee`), 101  
`BoolMask` (class in `astroNN.nn.layers`), 54

## C

`call()` (`astroNN.nn.layers.BoolMask` method), 55  
`call()` (`astroNN.nn.layers.ErrorProp` method), 45  
`call()` (`astroNN.nn.layers.FastMCInferenceMeanVar` method), 49  
`call()` (`astroNN.nn.layers.FastMCRepeat` method), 50  
`call()` (`astroNN.nn.layers.KLDivergenceLayer` method), 46  
`call()` (`astroNN.nn.layers.MCBatchNorm` method), 44  
`call()` (`astroNN.nn.layers.MCConcreteDropout` method), 40  
`call()` (`astroNN.nn.layers.MCDropout` method), 39  
`call()` (`astroNN.nn.layers.MCGaussianDropout` method), 43  
`call()` (`astroNN.nn.layers.MCSpatialDropout1D` method), 41  
`call()` (`astroNN.nn.layers.MCSpatialDropout2D` method), 42  
`call()` (`astroNN.nn.layers.PolyFit` method), 47  
`call()` (`astroNN.nn.layers.StopGrad` method), 53  
`call()` (`astroNN.nn.layers.TensorInput` method), 55  
`categorical_accuracy()` (in module `astroNN.nn.losses`), 36  
`categorical_crossentropy()` (in module `astroNN.nn.losses`), 30  
`chips_pix_info()` (in module `astroNN.apogee`), 98  
`chips_split()` (in module `astroNN.apogee`), 100  
`CNNBase` (class in `astroNN.models.base_cnn`), 72  
`combined_spectra()` (in module `astroNN.apogee`), 104  
`ConvVAEBase` (class in `astroNN.models.base_vae`), 76  
`custom_train_step()` (`astroNN.models.base_bayesian_cnn.BayesianCNNBase` method), 74  
`custom_train_step()` (`astroNN.models.base_vae.ConvVAEBase` method), 76

## E

`ErrorOnNaN` (class in `astroNN.nn.callbacks`), 58  
`ErrorProp` (class in `astroNN.nn.layers`), 45  
`evaluate()` (`astroNN.models.base_bayesian_cnn.BayesianCNNBase` method), 74  
`evaluate()` (`astroNN.models.base_cnn.CNNBase` method), 72  
`evaluate()` (`astroNN.models.base_vae.ConvVAEBase` method), 76

`extinction_correction()` (in module `astroNN.gaia`), 117

## F

`fakemag_to_absmag()` (in module `astroNN.gaia`), 115  
`fakemag_to_logsol()` (in module `astroNN.gaia`), 116  
`fakemag_to_mag()` (in module `astroNN.gaia`), 117  
`fakemag_to_parallax()` (in module `astroNN.gaia`), 115  
`fakemag_to_pc()` (in module `astroNN.gaia`), 115  
`FastMCInference` (class in `astroNN.nn.layers`), 51  
`FastMCInferenceMeanVar` (class in `astroNN.nn.layers`), 49  
`FastMCRepeat` (class in `astroNN.nn.layers`), 50  
`fit()` (`astroNN.models.base_bayesian_cnn.BayesianCNNBase` method), 74  
`fit()` (`astroNN.models.base_cnn.CNNBase` method), 73  
`fit()` (`astroNN.models.base_vae.ConvVAEBase` method), 76  
`fit_on_batch()` (`astroNN.models.base_bayesian_cnn.BayesianCNNBase` method), 75  
`fit_on_batch()` (`astroNN.models.base_cnn.CNNBase` method), 73  
`fit_on_batch()` (`astroNN.models.base_vae.ConvVAEBase` method), 76  
`flush()` (`astroNN.models.base_master_nn.NeuralNetMaster` method), 69

## G

`gaiadr2_parallax()` (in module `astroNN.gaia`), 111  
`gap_delete()` (in module `astroNN.apogee`), 99  
`get_config()` (`astroNN.models.base_master_nn.NeuralNetMaster` method), 69  
`get_config()` (`astroNN.nn.layers.BoolMask` method), 55  
`get_config()` (`astroNN.nn.layers.ErrorProp` method), 45  
`get_config()` (`astroNN.nn.layers.FastMCInference` method), 52  
`get_config()` (`astroNN.nn.layers.FastMCInferenceMeanVar` method), 49  
`get_config()` (`astroNN.nn.layers.FastMCRepeat` method), 50  
`get_config()` (`astroNN.nn.layers.KLDivergenceLayer` method), 46  
`get_config()` (`astroNN.nn.layers.MCBatchNorm` method), 44  
`get_config()` (`astroNN.nn.layers.MCConcreteDropout` method), 40  
`get_config()` (`astroNN.nn.layers.MCDropout` method), 39  
`get_config()` (`astroNN.nn.layers.MCGaussianDropout` method), 43

- get\_config() (*astroNN.nn.layers.MCSpatialDropout1D* method), 41
- get\_config() (*astroNN.nn.layers.MCSpatialDropout2D* method), 42
- get\_config() (*astroNN.nn.layers.PolyFit* method), 47
- get\_config() (*astroNN.nn.layers.StopGrad* method), 53
- get\_config() (*astroNN.nn.layers.TensorInput* method), 56
- get\_layer() (*astroNN.models.base\_master\_nn.NeuralNetMaster* method), 69
- get\_weights() (*astroNN.models.base\_master\_nn.NeuralNetMaster* method), 69
- ## H
- has\_model (*astroNN.models.base\_master\_nn.NeuralNetMaster* property), 69
- hessian() (*astroNN.models.base\_master\_nn.NeuralNetMaster* method), 69
- ## I
- input\_shape (*astroNN.models.base\_master\_nn.NeuralNetMaster* property), 70
- intpow\_avx2() (in module *astroNN.nn*), 61
- ## J
- jacobian() (*astroNN.models.base\_master\_nn.NeuralNetMaster* method), 70
- jacobian\_latent() (*astroNN.models.base\_vae.ConvVAEBase* method), 77
- ## K
- kl\_divergence() (in module *astroNN.nn.numpy*), 61
- KLDivergenceLayer (class in *astroNN.nn.layers*), 46
- ## L
- l1() (in module *astroNN.nn.numpy*), 62
- l2() (in module *astroNN.nn.numpy*), 62
- load\_allstar\_dr5() (in module *astroNN.lamost*), 110
- load\_apogee\_distances() (in module *astroNN.datasets*), 107
- load\_apogee\_rc() (in module *astroNN.datasets.apogee*), 106
- load\_folder() (in module *astroNN.models*), 79
- logsol\_to\_absmag() (in module *astroNN.gaia*), 117
- logsol\_to\_fakemag() (in module *astroNN.gaia*), 116
- ## M
- mag\_to\_absmag() (in module *astroNN.gaia*), 114
- mag\_to\_fakemag() (in module *astroNN.gaia*), 114
- magic\_correction\_term() (in module *astroNN.nn.losses*), 23
- MCBatchNorm (class in *astroNN.nn.layers*), 44
- MCConcreteDropout (class in *astroNN.nn.layers*), 40
- MCDropout (class in *astroNN.nn.layers*), 39
- MCGaussianDropout (class in *astroNN.nn.layers*), 43
- MCSpatialDropout1D (class in *astroNN.nn.layers*), 41
- MCSpatialDropout2D (class in *astroNN.nn.layers*), 42
- mean\_absolute\_error() (in module *astroNN.nn.losses*), 24
- mean\_absolute\_error() (in module *astroNN.nn.numpy*), 62
- mean\_absolute\_percentage\_error() (in module *astroNN.nn.losses*), 28
- mean\_absolute\_percentage\_error() (in module *astroNN.nn.numpy*), 63
- mean\_error() (in module *astroNN.nn.losses*), 25
- mean\_percentage\_error() (in module *astroNN.nn.losses*), 29
- mean\_squared\_error() (in module *astroNN.nn.losses*), 23
- mean\_squared\_logarithmic\_error() (in module *astroNN.nn.losses*), 28
- median\_absolute\_error() (in module *astroNN.nn.numpy*), 63
- median\_absolute\_percentage\_error() (in module *astroNN.nn.numpy*), 63
- module
- astroNN.apogee*, 96
  - astroNN.apogee.chips*, 96
  - astroNN.apogee.downloader*, 102
  - astroNN.config*, 15
  - astroNN.datasets.galaxy10*, 3
  - astroNN.datasets.galaxy10sdss*, 7
  - astroNN.gaia*, 110
  - astroNN.gaia.downloader*, 110
  - astroNN.lamost*, 108
  - astroNN.models*, 67
  - astroNN.models.apogee\_models*, 120, 125, 133, 140, 142, 147, 149
  - astroNN.neuralode*, 65
  - astroNN.neuralode.odeint*, 65
  - astroNN.nn*, 57, 60
  - astroNN.nn.callbacks*, 57
  - astroNN.nn.layers*, 38
  - astroNN.nn.losses*, 22
  - astroNN.nn.metrics*, 22
  - astroNN.nn.numpy*, 61
  - astroNN.nn.utilities*, 57
  - astroNN.nn.utilities.normalizer*, 58
  - mse\_lin\_wrapper*() (in module *astroNN.nn.losses*), 26
  - mse\_var\_wrapper*() (in module *astroNN.nn.losses*), 26
- ## N
- NeuralNetMaster (class in *astroNN.models.base\_master\_nn*), 68

## O

`odeint()` (in module `astroNN.neuralode.odeint`), 65

`output_shape` (`astroNN.models.base_master_nn.NeuralNetMaster` property), 71

## P

`plot_dense_stats()` (`astroNN.models.base_master_nn.NeuralNetMaster` method), 71

`plot_model()` (`astroNN.models.base_master_nn.NeuralNetMaster` method), 71

`PolyFit` (class in `astroNN.nn.layers`), 47

`predict()` (`astroNN.models.base_bayesian_cnn.BayesianCNNBase` method), 75

`predict()` (`astroNN.models.base_cnn.CNNBase` method), 73

`predict()` (`astroNN.models.base_vae.ConvVAEBase` method), 77

`predict_decoder()` (`astroNN.models.base_vae.ConvVAEBase` method), 77

`predict_encoder()` (`astroNN.models.base_vae.ConvVAEBase` method), 78

`pseudo_continuum()` (in module `astroNN.lamost`), 109

`pylab_style()` (in module `astroNN.shared.matplotlib`), 22

## R

`recompile()` (`astroNN.models.base_bayesian_cnn.BayesianCNNBase` method), 75

`recompile()` (`astroNN.models.base_cnn.CNNBase` method), 74

`recompile()` (`astroNN.models.base_vae.ConvVAEBase` method), 78

`reduce_var()` (in module `astroNN.nn`), 60

`relu()` (in module `astroNN.nn.numpy`), 64

`robust_binary_crossentropy()` (in module `astroNN.nn.losses`), 34

`robust_categorical_crossentropy()` (in module `astroNN.nn.losses`), 32

`robust_mse()` (in module `astroNN.nn.losses`), 25

## S

`save()` (`astroNN.models.base_master_nn.NeuralNetMaster` method), 71

`save_weights()` (`astroNN.models.base_master_nn.NeuralNetMaster` method), 71

`savefile()` (`astroNN.nn.callbacks.VirtualCSVLogger` method), 57

`sigmoid()` (in module `astroNN.nn.numpy`), 64

`sigmoid_inv()` (in module `astroNN.nn.numpy`), 64

`StarNet2017` (class in `astroNN.models.apogee_models`), 149

`StopGrad` (class in `astroNN.nn.layers`), 53

`summary()` (`astroNN.models.base_master_nn.NeuralNetMaster` method), 72

## T

`TensorInput` (class in `astroNN.nn.layers`), 55

`tgas()` (in module `astroNN.gaia`), 111

`tgas_load()` (in module `astroNN.gaia`), 112

`transfer_weights()` (`astroNN.models.base_master_nn.NeuralNetMaster` method), 72

## U

`uses_learning_phase` (`astroNN.models.base_master_nn.NeuralNetMaster` property), 72

## V

`VirtualCSVLogger` (class in `astroNN.nn.callbacks`), 57

`visit_spectra()` (in module `astroNN.apogee`), 104

## W

`wavelength_solution()` (in module `astroNN.apogee`), 99

`wavelength_solution()` (in module `astroNN.lamost`), 109

## X

`xmatch()` (in module `astroNN.datasets.xmatch`), 119

## Z

`zeros_loss()` (in module `astroNN.nn.losses`), 38