SASNets Documentation

Release 0.1a

Christopher Wang

Jan 10, 2018

Contents

Contents:

CHAPTER 1

SASNets

SASNets is a neural network implementation that classifies 1D SANS data as one of 71 SANS shape models, based on SASView and SASModels.

1.1 Features

- **Cross-platform:** Fully written in Python, SASNets will run on any platform that has a working Python distribution.
- **Extensible:** SASNets uses the standard Keras and Tensorflow libraries, so it is easy to build upon and improve.
- **Multifunction:** While SASNets was originally developed for SANS data, it can be applied to SAXS and non-SAS datasets.

1.2 Goal

Streamline the process of SANS and SAXS data analysis through the use of a convolutional neural network (CNN).

1.3 Motivation

Manual analysis of SAS data is difficult to do, especially with complex crystals and solutions. Convolutional Neural Networks have been shown to have great success on complex image recognition and retrieval tasks. We implement a novel combination of a convolutional neural network with 1D SAS data. The current iteration of the program was aimed at demonstrating the potential that CNNs could have on SAS fitting. Project extensions in the future will include expanding to 2D, tuning the network for even higher accuracy, and incorporating automatic bumps parameter fitting.

1.4 Results

SASNets has been able to achieve a prediction accuracy of between 40% and 50% on the 71 model set. We also demonstrate that the network is able to distinguish superclusters or types of models, for example cylinders or spheres.

1.5 More Information

For a more in-depth exploration of this project, please see `this poster` or `this presentation`.

A full size version of the dendrogram can be found `here <source/dendrogram.pdf`.

1.6 License

1.7 Contact

Chris Wang was the main author of the program, and can be reached at com *period* icloud *asperand* chrwang0, backwards. The project advisors were Paul Kienzle and William Ratcliff of NCNR.

All code can be found online at https://github.com/scattering/sasnets. If you have any questions or contributions, file an issue on Github, or send me(Chris) an email.

1.8 Acknowledgements

This project benefited from the NSF-funded DANSE Project, DMR-0520547, SANS Subproject from the University of Tennessee Knoxville.

This project benefited from funding and material support from the NIST Centre for Neutron Research and the NSF-funded CHRNS.

SASView and SASModels were developed by an international team as part of the NSF-funded DANSE project. Their work can be found at https://github.com/sasview. The custom fork of SASModels used in this program can be found at https://github.com/chrwang/sasmodels.

Finally, thanks to all of the people who helped read, test, and critique the project.

CHAPTER 2

Installation

2.1 System Requirements

SASNets requires a system with a *nix style command line. Any Unix/Unix-like distributions will satisfy this (Linux, macOS, RedHat, etc.). On Windows, you will either need the WSL (untested, but should work) or a terminal system such as MinGW or Cygwin.

You'll need either Python 2.7 or 3.6 installed. While SASNets can theoretically run on any hardware, we recommend at least 4 physical cores, at least a GTX 970 or similar card, and at least 16 GB of RAM. On systems with lower specifications, training will be extremely slow, and may crash on memory-constrained systems. Consider training on AWS or Google Cloud instances if you cannot obtain such a system physically. Any performance metrics cited in these docs were, unless otherwise stated, run on a GTX 1080 Ti, Intel i7-7700, and 32 GB of DDR4.

2.2 Install Procedure

Installing SASNets is relatively easy. First, clone the Github repository https://github.com/scattering/SASNets. We recommend installing SASNets in a virtualenv to keep its packages separate from others. Do a

```
python setup.py install
```

which will install the SASNets and related dependencies. There are optional features (specified as extras in setup.py) which you can install to gain additional functionality. Currently, this consists of:

- **ruamel.yaml**: Used for better json parsing. Only relevant for sequential read from .json formatted files.
- **bumps**: Used for fitting data to theory models output by the neural network. Currently unimplemented.
- **Sphinx**: Used for building documentation locally. The docs are also online at https://sasnets.readthedocs.io.
- **seaborn**: Used in some matplotlib plots for more colourful output.

Note that setup.py installs the pip version of Tensorflow, which may or may not come with the features that you need. If you would like Nvidia GPU support, install tensorflow-gpu from pip and comment out tensorflow from setup.py. If you would like more advanced features such as OpenCL on CPUs and Intel GPUs or Google Cloud Platform support, compile from source. Note that Nvidia support for Tensorflow on macOS is deprecated.

It is highly recommended that you use the PostgreSQL version of SASNets, as this uses significantly less memory and is much faster. This requires you to have PostgreSQL installed. Consult your OS package manager documentation for how to do so.

If you have any problems, file an issue on Github or contact the developers directly.

CHAPTER 3

sasnets package

3.1 Subpackages

3.1.1 sasnets.util package

Submodules

sasnets.util.logsql module

`sasnets.util.logsql.`**`main`**`(`*args*`)`

Main function. Args should conform to the argparse args specified.

Parameters **`args`** – Arguments from the command line

Returns None

sasnets.util.p2sql module

Utility script that uploads data files to a python script.

Reads files generated by sasmodels and uploads the individual data sessions to a postgres database running on localhost. Slow.

`sasnets.util.p2sql.`**`main`**`(`*args*`)`

Main function. Args should conform to the argparse args specified.

Parameters **`args`** – Arguments from the command line

Returns None

sasnets.util.rnames module

`sasnets.util.rnames.`**`main`**`(`*args*`)`

`sasnets.util.rnames.`**`rnames`**`(`*path, p='_all_'*`)`

Method to collect all filenames of models from a directory.

rnames reads all files contained in path that match the regex p, and outputs the model name to a new file named name. If name exists, it is overwritten.

Parameters

- **`path`** – A string representing a filepath.
- **`p`** – A regex to match files to.

Returns None

sasnets.util.tosql module

Utility script that uploads data files to a python script.

Reads files generated by sasmodels and uploads the individual data sessions to data.db in a table specified from a command line. Slow.

`sasnets.util.tosql.`**`main`**(*args*)

Main method. args come from system command line; should conform to argparse arguments.

Parameters **`args`** – Command line args

Returns None

sasnets.util.utils module

Various small utility functions that are reused throughout the program.

`sasnets.util.utils.`**`inepath`**(*pathname*)

Returns a normalised path given a path. Checks if the path exists and creates the path if it does not. If a directory is specified, appends the current time as the filename.

Parameters **`pathname`** – The pathname to process.

Returns A normalised path, or None.

`sasnets.util.utils.`**`plot`**(*q*, *i_q*)

Method to plot Q vs I(Q) data for testing and verification purposes.

Parameters

- **`q`** – List of Q values
- **`i_q`** – List of I values

Returns None

Module contents

3.2 Submodules

3.3 sasnets.analysis module

File used for analysis of SASNet networks using various techniques, including dendrograms and confusion matrices.

`sasnets.analysis.`**`cpredict`**(*model*, *x*, *l=71*, *pl=5000*)

Runs a Keras model to create a confusion matrix.

Parameters

- **`model`** – Model to use.
- **`x`** – A list of x values to predict on.
- **`l`** – The number of input models.

- **`pl`** – The number of data iterations per model.

Returns A confusion matrix of percentages

`sasnets.analysis.`**`dcluster`**(*model*, *x*, *names*)

Displays a dendrogram clustering based on the confusion matrix.

Parameters

- **`model`** – The model to predict on.
- **`x`** – A list of x values to predict on.
- **`names`** – List of all model names.

Returns The dendrogram object.

`sasnets.analysis.`**`fit`**(*mn*, *q*, *iq*)

Fit resulting data using bumps server. Currently unimplemented.

Parameters

- **`mn`** – Model name.
- **`q`** – List of q values.
- **`iq`** – List of I(q) values.

Returns Bumps fit.

`sasnets.analysis.`**`load_from`**(*path*)

Loads a model from the specified path.

Parameters **`path`** – Relative or absolute path to the .h5 model file

Returns The loaded model.

`sasnets.analysis.`**`main`**(*args*)

Main method. Called from command line; uses argparse.

Parameters **`args`** – Arguments from command line.

Returns None.

`sasnets.analysis.`**`predict`**(*model*, *x*, *names*, *num=5*)

Runs a Keras model to predict based on input.

Parameters

- **`model`** – The model to use.
- **`x`** – The x inputs to predict from.
- **`names`** – A list of all model names.
- **`num`** – The top num probabilities and models will be printed.

Returns None

`sasnets.analysis.`**`predict_and_val`**(*model*, *x*, *y*, *names*)

Runs the model on the input datasets and compares the results with the correct labels provided from y.

Parameters

- **`model`** – The model to evaluate.
- **`x`** – List of x values to predict on.
- **`y`** – List of y values to predict on.
- **`names`** – A list of all possible model names.

Returns Two lists, il and nl, which are the indices of the model and its proper name respectively.

`sasnets.analysis.`**`rpredict`**(*model*, *x*, *names*)
Same as predict, but outputs names only.

Parameters

- **`model`** – The model to use.
- **`x`** – List of x to predict on.
- **`names`** – List of all model names.

Returns List of predicted names.

`sasnets.analysis.`**`tcluster`**(*model*, *x*, *names*)
Displays a t-SNE cluster coloured by the model predicted labels.

Parameters

- **`model`** – Model to use.
- **`x`** – List of x values to predict on.
- **`names`** – List of all model names.

Returns The tSNE object that was plotted.

3.4 sasnets.hyp module

`sasnets.hyp.`**`data`**()

`sasnets.hyp.`**`model`**(*xtrain*, *ytrain*, *xtest*, *ytest*)

3.5 sasnets.sas_io module

Collection of utility IO functions used in SASNet. Contains the read from disk functions as well as the SQL generator.

`sasnets.sas_io.`**`read_h`**(*l*)
Read helper for parallel read.

Parameters **`l`** – A list of filenames to read from.

Returns Three lists, Q, IQ, and Y, corresponding to Q data, I(Q) data, and model labels respectively.

`sasnets.sas_io.`**`read_parallel_1d`**(*path*, *pattern='_eval_'*)
Reads all files in the folder path. Opens the files whose names match the regex pattern. Returns lists of Q, I(Q), and ID. Path can be a relative or absolute path. Uses Pool and map to speed up IO. WIP. Uses an excessive amount of memory currently. It is recommended to use sequential on systems with less than 16 GiB of memory.

Calling parallel on 69 150k line files, a gc, and parallel on 69 5k line files takes around 70 seconds. Running sequential on both sets without a gc takes around 562 seconds. Parallel peaks at 15 + GB of memory used with two file reading threads. Sequential peaks at around 7 to 10 GB. Use at your own risk. Be prepared to kill the threads and/or press the reset button.

Assumes files contain 1D data.

Parameters

- **`path`** – Path to the directory of files to read from.
- **`pattern`** – A regex. Only files matching this regex are opened.

`sasnets.sas_io.`**`read_seq_1d`**(*path*, *pattern='_eval_'*, *typef='aggr'*, *verbosity=False*)

Reads all files in the folder path. Opens the files whose names match the regex pattern. Returns lists of Q, I(Q), and ID. Path can be a relative or absolute path. Uses a single thread only. It is recommended to use `read_parallel_1d()`, except in hyperopt, where map() is broken.

typef is one of 'json' or 'aggr'. JSON mode reads in all and only json files in the folder specified by path. aggr mode reads in aggregated data files. See sasmodels/generate_sets.py for more about these formats.

Assumes files contain 1D data.

Parameters

- **`path`** – Path to the directory of files to read from.
- **`pattern`** – A regex. Only files matching this regex are opened.
- **`typef`** – Type of file to read (aggregate data or json data).
- **`verbosity`** – Controls the verbosity of output.

`sasnets.sas_io.`**`sql_dat_gen`**(*dname*, *mname*, *dbname='sas_data'*, *host='127.0.0.1'*, *user='sasnets'*, *encoder=None*)

A Pythonic generator that gets its data from a PostgreSQL database. Yields a (iq, diq) list and a label list.

Parameters

- **`dname`** – The data table name to connect to.
- **`mname`** – The metadata table name to connect to.
- **`dbname`** – The database name.
- **`host`** – The database host.
- **`user`** – The username to connect as.
- **`encoder`** – LabelEncoder for transforming labels to categorical ints.

Returns None

3.6 sasnets.sasnet module

3.7 Module contents

CHAPTER 4

Working With Custom Datasets

SASNets is designed to be relatively easy to extend to any kind of data. This page describes how to use different SANS and SAXS datasets, as well as completely different datasets.

4.1 SASModels

SASNets makes use of data generated by a custom fork of SASModels, which contains the custom models used inside SASView. Generating your own SANS data is easy: just run

```
./gen_models.sh <models> <number> <dim> <npoints> <cutoff> <precision>
```

where:

- models is the models that are to be run, or a class of models.
- number is the number of distinct models to generate.
- dim is the dimensionality of data (1D or 2D).
- npoints is the number of points per model.
- cutoff is the polydisperse cutoff, or ‘mono’.
- precision is the precision level.

More information on these parameters can be found at http://sasview.org/docs.

The gen_models.sh script is short:

```
1 #!/bin/bash
2 sasview=( ../sasview/build/lib.* )
3 sep=$(python -c "import os;print(os.pathsep)")
4 PYTHONPATH=../bumps${sep}../periodictable${sep}$sasview
5 export PYTHONPATH
6 PYOPENCL_COMPILER_OUTPUT=1; export PYOPENCL_COMPILER_OUTPUT
7 PYOPENCL_CTX=2; export PYOPENCL_CTX
8 python -m sasmodels.generate_sets $*
```

It first sets the location of SASView, bumps and periodictable, which are used in generating data for models (lines 2 through 5). It then enables pyopencl compiler output, which will print out warnings when compiling models for the CPU. With CTX, this selects a OpenCL device to compute on. This is an index from

```
pyopencl.create_some_context()
```

The exact number can be determined by running this command in a Python shell. The Intel CPU is typically 0, the Intel GPU 1, and the discrete unit (AMD or Nvidia) 2. It is recommended that you use the discrete GPU if possible, as it is many times faster than the CPU/integrated graphics. It is known that some models are broken on certain hardware devices. If a model fails to produce data with "Maths error, bad model", try forcing SASModels to build using a different device, if you have one.

An example call to gen_models for training would be

```
./gen_models.sh all 15000 1D 100 mono double
```

These results are saved (by default) to a PostgreSQL table named "train_data" in a database named "sas_data", with the username and password "sasnets". train_data's schema is:

Column	Type
id	integer
iq	numeric[]
diq	numeric[]
model	text

We recommend at least 10,000 data points per model for training a neural network for research, and at least 20,000 to 25,000 for production systems. With 15,000 points per model, an epoch with batch size 5 on approximately 800,000 points takes approximately 450 seconds to run. During training, you can optionally specify evaluation data, which gives more accurate statistics on the model than using training data would. Changing line 188 to

```
"INSERT INTO eval_data (iq, diq, model)....
```

and rerunning the script will randomly generate data and put it in the eval_data table, which has the same columns as the train_data table. We recommend 2500 evaluation points per model.

Depending on the instrument configuration that you are generating data for, parameters within generate_sets.py may have to be edited. Lines 283 and 284 contain the most relevant parts, namely the Q-range and noise. You can optionally read *Q* and *dQ* ranges from files and use these instead, which is done in lines 285 through 287. The data object produced by `make_data` is an object with various instance variables depending on the options passed to the function. *q* is represented as *data.x* and *dq* is represented by *data.dx*.

Python Module Index

Index