deepnog

Release 1.1.0

Lukas Gosch, Roman Feldbauer

Aug 28, 2020

GETTING STARTED

`deepnog` is a Python package for predicting protein orthologous groups with deep networks.

CHAPTER

ONE

INSTALLATION

1.1 Installation from PyPI

The current release of `deepnog` can be installed from PyPI:

```
pip install deepnog
```

For typical use cases, and quick start, this is sufficient.

1.2 Dependencies and model files

All package dependencies of `deepnog` are automatically installed by `pip`. We also require model files (networks parameters/weights), which are too large for GitHub/PyPI. These are hosted on separate servers, and downloaded automatically by `deepnog`, when required. By default, models are cached in *$HOME/deepnog_data/*.

You can change this path by setting the DEEPNOG_DATA environment variable.

```
DEEPNOG_DATA="/custom/path/models" deepnog sequences.fa
```

1.3 Installation from source

You can always grab the latest version of `deepnog` directly from GitHub:

```
cd install_dir
git clone git@github.com:VarIr/deepnog.git
cd deepnog
pip install -e .
```

This is the recommended approach, if you want to contribute to the development of `deepnog`.

1.4 Supported platforms

`deepnog` currently supports all major operating systems:

- Linux
- MacOS X
- Windows

CHAPTER

TWO

QUICK START EXAMPLE

Users of `deepnog` typically want to

1. ...
2. ...
3. ...

The following example shows all these steps for an example dataset. Please make sure you have installed `deepnog` (installation instructions).

First, we load the dataset and inspect its size.

```
from deepnog import predict
...
```

```
deepnog input.fa --out prediction.csv -db eggNOG5 --tax 2
```

CHAPTER

THREE

USER GUIDE

Welcome to `deepnog`! Here we describe the core functionality of the package (...), and provide several usage examples.

3.1 Concepts

concepts

3.2 Examples

examples

CHAPTER

FOUR

API DOCUMENTATION

This is the API documentation for `deepnog`.

4.1 DeepNOG

DeepNOG is a deep learning based command line tool which predicts the protein families of given protein sequences based on pretrained neural networks.

The main module of this tool is defined in deepnog.py. For details about the usage of the tool, the reader is referred to the documentation as well as deepnog.py.

4.2 deepnog.client

Author: Lukas Gosch

Date: 2019-10-18

Usage: python client.py –help

Description:

> Provides the `deepnog` command line client and entry point for users.
>
> DeepNOG predicts protein families/orthologous groups of given protein sequences with deep learning.
>
> File formats supported: Preferred: FASTA DeepNOG supports protein sequences stored in all file formats listed in https://biopython.org/wiki/SeqIO but is tested for the FASTA-file format only.
>
> Architectures supported:
>
> **Databases supported:**
>
> - eggNOG 5.0, taxonomic level 1 (root)
> - eggNOG 5.0, taxonomic level 2 (bacteria)

`deepnog.client.`**`get_parser`**`()`

Creates a new argument parser.

> **Returns** **parser** – ArgumentParser object to parse program arguments.
>
> **Return type** ArgumentParser

`deepnog.client.`**`main`**`()`

DeepNOG command line tool.

`deepnog.client.`**`start_prediction`**`(`*`args`*`)`

4.3 deepnog.dataset

Author: Lukas Gosch

Date: 2019-10-03

Description:

Dataset classes and helper functions for usage with deep network models written in PyTorch.

class deepnog.dataset.**ProteinDataset**(*file*, *f_format='fasta'*)
Bases: torch.utils.data.dataset.IterableDataset

Protein dataset holding the proteins to classify.

Does not load and store all proteins from a given sequence file but only holds an iterator to the next sequence to load.

Thread safe class allowing for multi-worker loading of sequences from a given datafile.

Parameters

- **file** (*str*) – Path to file storing the protein sequences.
- **f_format** (*str*) – File format in which to expect the protein sequences. Must be supported by Biopython's Bio.SeqIO class.

class deepnog.dataset.**ProteinIterator**(*file_*, *aa_vocab*, *f_format*, *n_skipped: Union[int, deepnog.sync.SynchronizedCounter] = 0*, *num_workers=1*, *worker_id=0*)
Bases: object

Iterator allowing for multiprocess data loading of a sequence file.

ProteinIterator is a wrapper for the iterator returned by Biopython's Bio.SeqIO class when parsing a sequence file. It specifies custom __next__() method to support single- and multi-process data loading.

In the single-process loading case, nothing special happens, the ProteinIterator sequentially iterates over the data file. In the end, it informs the main module about the number of skipped sequences (due to empty ids) through setting a global variable in the main module.

In the multi-process loading case, each ProteinIterator loads a sequence and then skips the next few sequences dedicated to the other workers. This works by each worker skipping num_worker - 1 data samples for each call to __next__(). Furthermore, each worker skips worker_id data samples in the initialization. At the end of the workers lifetime, it sends the number of skipped sequences back to the main process through a pipe the main process created.

The ProteinIterator class also makes sure that a unique ID is set for each SeqRecord obtained from the data-iterator. This allows unambiguous handling of large protein datasets which may have duplicate IDs from merging multiple sources or may have no IDs at all. For easy and efficient sorting of batches of sequences as well as for direct access to the original IDs, the index is stored separately.

Parameters

- **file** (*str*) – Path to sequence file, from which an iterator over the sequences will be created with Biopython's Bio.SeqIO.parse() function.
- **aa_vocab** (*dict*) – Amino-acid vocabulary mapping letters to integers
- **f_format** (*str*) – File format in which to expect the protein sequences. Must be supported by Biopython's Bio.SeqIO class.
- **num_workers** (*int*) – Number of workers set in DataLoader or one if no workers set. If bigger or equal to two, the multi-process loading case happens.

- **worker_id** (*int*) – ID of worker this iterator belongs to

deepnog.dataset.**collate_sequences**(*batch*, *zero_padding=True*)
Collate and zero-pad encoded sequence.

Parameters

- **batch** (*list[namedtuple] or namedtuple*) – Batch of protein sequences to classify stored as a namedtuple-class sequence (see ProteinDataset).
- **zero_padding** (*bool*) – If True, zero-pads protein sequences through appending zeros until every sequence is as long as the longest sequences in batch. If False raise NotImplementedError.

Returns **batch** – Input batch zero-padded and stored in namedtuple-class collated_sequences.

Return type namedtuple

class deepnog.dataset.**collated_sequences**(*indices*, *ids*, *sequences*)
Bases: tuple

count(*value*, */*)
Return number of occurrences of value.

property ids
Alias for field number 1

index(*value*, *start=0*, *stop=9223372036854775807*, */*)
Return first index of value.

Raises ValueError if the value is not present.

property indices
Alias for field number 0

property sequences
Alias for field number 2

deepnog.dataset.**consume**(*iterator*, *n=None*)
Advance the iterator n-steps ahead. If n is None, consume entirely.

Function from Itertools Recipes in official Python 3.7.4. docs.

deepnog.dataset.**gen_amino_acid_vocab**(*alphabet=None*)
Create vocabulary for protein sequences.

A vocabulary is defined as a mapping from the amino-acid letters in the alphabet to numbers. As this mapping is aware of zero-padding, it maps the first letter in the alphabet to 1 instead of 0.

Parameters **alphabet** (*str*) – Alphabet to use for vocabulary. If None, use 'ACDEFGHIKLMNPQRSTVWYBXZJUO' (equivalent to deprecated Biopython's ExtendedIUPACProtein).

Returns **vocab** – Mapping of amino acid characters to numbers.

Return type dict

4.4 deepnog.inference

Author: Roman Feldbauer

Date: 2020-02-19

Description:

Predict orthologous groups of protein sequences.

deepnog.inference.**load_nn**(*architecture*, *model_dict*, *device='cpu'*)

Import NN architecture and set loaded parameters.

Parameters

- **architecture** (*str*) – Name of neural network module and class to import.
- **model_dict** (*dict*) – Dictionary holding all parameters and hyper-parameters of the model.
- **device** (*[str, torch.device]*) – Device to load the model into.

Returns model – Neural network object of type architecture with parameters loaded from model_dict and moved to device.

Return type torch.nn.Module

deepnog.inference.**predict**(*model*, *dataset*, *device='cpu'*, *batch_size=16*, *num_workers=4*, *verbose=3*)

Use model to predict zero-indexed labels of dataset.

Also handles communication with ProteinIterators used to load data to log how many sequences have been skipped due to having empty sequence ids.

Parameters

- **model** (*nn.Module*) – Trained neural network model.
- **dataset** (ProteinDataset) – Data to predict protein families for.
- **device** (*[str, torch.device]*) – Device of model.
- **batch_size** (*int*) – Forward batch_size proteins through neural network at once.
- **num_workers** (*int*) – Number of workers for data loading.
- **verbose** (*int*) – Define verbosity.

Returns

- **preds** (*torch.Tensor, shape (n_samples,)*) – Stores the index of the output-node with the highest activation
- **confs** (*torch.Tensor, shape (n_samples,)*) – Stores the confidence in the prediction
- **ids** (*list[str]*) – Stores the (possible empty) protein labels extracted from data file.
- **indices** (*list[int]*) – Stores the unique indices of sequences mapping to their position in the file

4.5 deepnog.io

Author: Roman Feldbauer

Date: 2020-02-19

Description:

Input/output helper functions

deepnog.io.**create_df**(*class_labels*, *preds*, *confs*, *ids*, *indices*, *threshold=None*, *verbose=3*)

Creates one dataframe storing all relevant prediction information.

The rows in the returned dataframe have the same order as the original sequences in the data file. First column of the dataframe represents the position of the sequence in the datafile.

Parameters

- **class_labels** (*list*) – Store class name corresponding to an output node of the network.
- **preds** (*torch.Tensor, shape (n_samples,)*) – Stores the index of the output-node with the highest activation
- **confs** (*torch.Tensor, shape (n_samples,)*) – Stores the confidence in the prediction
- **ids** (*list[str]*) – Stores the (possible empty) protein labels extracted from data file.
- **indices** (*list[int]*) – Stores the unique indices of sequences mapping to their position in the file
- **threshold** (*int*) – If given, prediction labels and confidences are set to '' if confidence in prediction is not at least threshold.
- **verbose** (*int*) – If bigger 0, outputs warning if duplicates detected.

Returns **df** – Stores prediction information about the input protein sequences. Duplicates (defined by their sequence_id) have been removed from df.

Return type pandas.DataFrame

deepnog.io.**get_data_home**(*data_home: str = None*) → pathlib.Path

Return the path of the deepnog data dir.

This folder is used for large files that cannot go into the Python package on PyPI etc. For example, the network parameters (weights) files may be larger than 100MiB. By default the data dir is set to a folder named 'deepnog_data' in the user home folder. Alternatively, it can be set by the 'DEEPNOG_DATA' environment variable or programmatically by giving an explicit folder path. If the folder does not already exist, it is automatically created.

Parameters **data_home** (*str | None*) – The path to deepnog data dir.

Notes

Adapted from SKLEARN_DATAHOME.

deepnog.io.**get_weights_path**(*database: str*, *level: str*, *architecture: str*, *data_home=None*, *download_if_missing=True*) → pathlib.Path

Get path to neural network weights.

This is a path on local storage. If the corresponding files are not present, download from remote storage. The default remote URL can be overridden by setting the environment variable DEEPNOG_REMOTE.

Parameters

- **database** (*str*) – The orthologous groups database. Example: eggNOG5
- **level** (*str*) – The taxonomic level within the database. Example: 2 (for bacteria)
- **architecture** (*str*) – Network architecture. Example: deepencoding
- **data_home** (*string, optional*) – Specify another download and cache folder for the weights. By default all deepnog data is stored in '~/deepnog_data' subfolders.
- **download_if_missing** (*boolean, default=True*) – If False, raise a IOError if the data is not locally available instead of trying to download the data from the source site.

Returns **weights_path** – Path to file of network weights

Return type Path

4.6 deepnog.models

Description:

Network definitions (PyTorch modules)

4.7 deepnog.sync

Author: Roman Feldbauer

Date: 2020-02-19

Description:

Parallel processing helpers

class deepnog.sync.**SynchronizedCounter**(*init: int = 0*)

Bases: object

A multiprocessing-safe counter.

Parameters **init** (*int, optional*) – Counter starts at init (default: 0)

increment(*n=1*)

Obtain a lock before incrementing, since += isn't atomic.

Parameters **n** (*int, optional*) – Increment counter by n (default: 1)

increment_and_get_value(*n=1*) → int

Obtain a lock before incrementing, since += isn't atomic.

Parameters **n** (*int, optional*) – Increment counter by n (default: 1)

property value

4.8 deepnog.tests

Description:

deepnog unit tests

4.9 deepnog.utils

Author: Roman Feldbauer

Date: 2020-02-19

Description:

Various utility functions

deepnog.utils.**set_device**(*device*)

Set device (CPU/GPU) depending on user choice and availability.

Parameters **device** (*str*) – Device set by user as an argument to DeepNOG call.

Returns **device** – Object containing the device type to be used for prediction calculations.

Return type torch.device

CHAPTER

FIVE

CONTRIBUTING

deepnog is free open source software. Contributions from the community are highly appreciated. Even small contributions improve the software's quality.

Even if you are not familiar with programming languages and tools, you may contribute by filing bugs or any problems as a GitHub issue.

5.1 Git and branching model

We use *git* for version control (CVS), as do most projects nowadays. If you are not familiar with git, there are lots of tutorials on GitHub Guide. All the important basics are covered in the GitHub Git handbook.

Development of *deepnog* (mostly) follows this git branching model. We currently use one main branch: master. For any changes, a new branch should be created. This includes new feature, noncritical or critical bug fixes, etc.

5.2 Workflow

In case of large changes to the software, please first get in contact with the authors for coordination, for example by filing an issue. If you want to fix small issues (typos in the docs, obvious errors, etc.) you can - of course - directly submit a pull request (PR).

1. **Create a fork of *deepnog* in your GitHub account.** Simply click "Fork" button on https://github.com/VarIr/deepnog.
2. **Clone your fork on your computer.** $ `git clone git@github.com:YOUR-ACCOUNT-GOES-HERE/deepnog.git && cd deepnog`
3. **Add remote upstream.** $ `git remote add upstream git@github.com:VarIr/deepnog.git`
4. **Create feature/bugfix branch.** $ `git checkout -b bugfix123 master`
5. **Implement feature/fix bug/fix typo/...** Happy coding!
6. **Create a commit with meaningful message** If you only modified existing files, simply `$ git commit -am "descriptive message what this commit does (in present tense) here"`
7. **Push to GitHub** e.g. $ `git push origin featureXYZ`
8. **Create pull request (PR)** Git will likely provide a link to directly create the PR. If not, click "New pull request" on your fork on GitHub.

9. **Wait…** Several devops checks will be performed automatically (e.g. continuous integration (CI) with Travis, AppVeyor).

   The authors will get in contact with you, and may ask for changes.

10. **Respond to code review.** If there were issues with continuous integration, or the authors asked for changes, please create a new commit locally, and simply push again to GitHub as you did before. The PR will be updated automatically.

11. **Maintainers merge PR, when all issues are resolved.** Thanks a lot for your contribution!

5.3 Code style and further guidelines

- Please make sure all code complies with PEP 8
- All code should be documented sufficiently (functions, classes, etc. must have docstrings with general description, parameters, ideally return values, raised exceptions, notes, etc.)
- Documentation style is NumPy format.
- New code must be covered by unit tests using pytest.
- If you fix a bug, please provide regression tests (fail on old code, succeed on new code).
- It may be helpful to install *deepnog* in editable mode for development. When you have already cloned the package, switch into the corresponding directory, and

```
pip install -e .
```

  (don't omit the trailing period). This way, any changes to the code are reflected immediately. That is, you don't need to install the package each and every time, when you make changes while developing code.

5.4 Testing

In *deepnog*, we aim for high code coverage. As of Feb 2020, more than 95% of all code lines are visited at least once when running the complete test suite. This is primarily to ensure:

- correctness of the code (to some extent) and
- maintainability (new changes don't break old code).

Creating a new PR, ideally all code would be covered by tests. Sometimes, this is not feasible or only with large effort. Pull requests will likely be accepted, if the overall code coverage at least does not decrease.

Unit tests are automatically performed for each PR using CI tools online. This may take some time, however. To run the tests locally, you need *pytest* installed. From the deepnog directory, call

```
pytest deepnog/
```

to run all the tests. You can also restrict the tests to the subpackage you are working on, down to single tests. For example

```
pytest deepnog/tests/test_dataset.py --showlocals -v
```

only runs tests about datasets.

In order to check code coverage locally, you need the pytest-cov plugin.

```
pytest deepnog --cov=deepnog
```

CHAPTER

SIX

CHANGELOG

6.1 Next release

...

6.2 1.1.0 - 2020-02-28

6.2.1 Added

- EggNOG5 root (tax 1) prediction

6.2.2 Changed

- Package structure changed for higher modularity. This will require changes in downstream usages.
- Remove network weights from the repository, because files are too large for github and/or PyPI. `deepnog` automatically downloads these from CUBE servers, and caches them locally.
- More robust inter-process communication in data loading

6.2.3 Fixes

- Fix error on very short amino acid sequences
- Fix error on unrecognized symbols in sequences (stop codons etc.)
- Fix multiprocess data loading from gzipped files
- Fix type mismatch in deepencoding embedding layer (Windows only)

6.2.4 Maintenance

- Continuous integration on
  - Travis (Linux, MacOS)
  - AppVeyor (Windows)
- Codecov coverage reports
- LGTM code quality/security reports
- Documentation on ReadTheDocs
- Upload to PyPI, thus enabling `$ pip install deepnog`.

6.3 1.0.0 - 2019-10-18

The first release of `deepnog` to appear in this changelog. It already contains the following features:

- EggNOG5 bacteria (tax 2) prediction
- DeepEncoding architecture
- CPU and GPU support
- Runs on all major platforms (Linux, MacOS, Windows)

CHAPTER

SEVEN

GETTING STARTED

Get started with `deepnog` in a breeze. Find how to install the package and see all core functionality applied in a single quick start example.

CHAPTER

EIGHT

USER GUIDE

The User Guide introduces the main concepts of `deepnog`.

CHAPTER

NINE

API DOCUMENTATION

The API Documentation provides detailed information of the implemented methods. This information includes method descriptions, parameters, references, examples, etc. Find all the information about specific modules and functions of `deepnog` in this section.

CHAPTER

TEN

DEVELOPMENT

There are several possibilities to contribute to this free open source software. We highly appreciate all input from the community, be it bug reports or code contributions.

Source code, issue tracking, discussion, and continuous integration appear on our GitHub page.

CHAPTER

ELEVEN

WHAT'S NEW

To see what's new in the latest version of `deepnog`, have a look at the changelog.

CHAPTER

TWELVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

d

INDEX