DEEPNOG
DEEP LEARNING TOOL FOR FAST ORTHOLOGOUS GROUP PREDICTIONS OF PROTEINS

USER & DEVELOPER GUIDE

Lukas Gosch and Roman Feldbauer

CONTENTS

1 Introduction

DeepNOG is a command line tool written in Python 3. It uses trained neural networks for extremely fast protein homology predictions. In its current installation, it is based upon a neural network architecture called DeepEncoding trained on the root and bacterial level of the eggNOG 5.0 database (Huerta-Cepas et al. (2019)). Note: The research article may not distinguish between tool and network architecture for simplicity, and may refer to both of them as 'DeepNOG'.

1.1 How to Read this Report

Independent on if you plan to only use DeepNOG or extend it, it is recommended that you familiarize yourself with the basic workflow of DeepNOG as outlined in Section 1.2. After installing the tool as described in Section 2, to use DeepNOG to classify your custom proteins, see the user's guide in Section 3 for more details. If you plan to extend DeepNOG with other neural network architectures or different supported databases, see Section 4 for more details.

1.2 Basic Workflow

DeepNOG's workflow is visualized in the process diagram in figure 1.

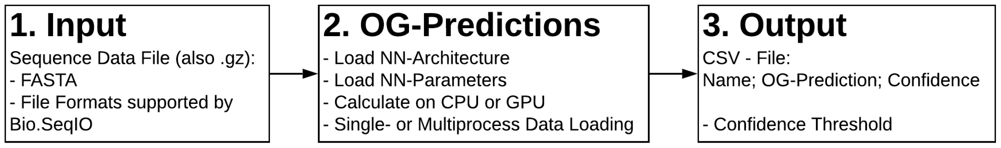
1. Input
Sequence Data File (also .gz):
- FASTA
- File Formats supported by Bio.SeqIO
2. OG-Predictions
- Load NN-Architecture
- Load NN-Parameters
- Calculate on CPU or GPU
- Single- or Multiprocess Data Loading
3. Output
CSV - File:
Name; OG-Prediction; Confidence
- Confidence Threshold

Figure 1: Workflow of DeepNOG to create orthologous group predictions based on input protein sequences. For a detailed explanation of the individual steps, the reader is referred to the description in the text.

1.2.1 Input Data

As an input DeepNOG expects a protein sequence file which can also be provided gzipped. It is tested for the FASTA file format but in general should support all file formats supported by the Bio.SeqIO module of Biopython. Following the conventions in the bioinformatics field, protein sequences, given no IDs in the input data file, are skipped and not used for the following prediction phase. Furthermore, if two sequences in the input data file have the same associated ID, only the sequence encountered first in the input data file will be kept and all others discarded before the output file is created. The user will be notified if such cases are encountered.

1.2.2 Prediction Phase

In the prediction phase, DeepNOG loads a predefined neural network and the corresponding trained weights (defaults to DeepEncoding trained on eggNOG 5.0 (bacterial level)). Then it performs the prediction through forwarding the input sequences through the network performing the calculations either on a CPU or GPU. DeepNOG offers single-process data loading aimed for calculations on a single CPU core to produce as little overhead as possible. Additionally, it offers parallel multiprocess data loading aimed for very fast GPU calculations. This is, to provide the GPU with data following up the previous forward pass fast enough such that the GPU does not experience idling. In its default parametrization, DeepNOG is optimized for single core CPU calculations, for details on how to best exploit GPUs for orthologous group predictions using DeepNOG, the reader is referred to the advanced Section 3.3 in this user's guide.

1.2.3 Output Data

As an output DeepNOG generates a CSV file which consists of three columns. First, the unique names or IDs of the proteins extracted from the sequence file, the second column corresponds to the OG-predictions and in the third column the confidence of the neural network in the prediction is stored. Each neural network model has the possibility to define a prediction confidence threshold below which, the neural network's output layer is treated as having predicted that the input protein sequence is not associated to any orthologous group in the model. Therefore, if the highest prediction confidence for any OG for a given input protein sequence is below this threshold, the prediction is left empty. Per default, using DeepEncoding on eggNOG 5.0, the prediction confidence threshold is set to a strict 99%.

2 Installation

2.1 Required Packages

DeepNOG is requires[1] the following packages:

- Python 3.7.4
- PyTorch 1.2.0
- NumPy 1.16.4
- pandas 0.25.1
- Biopython 1.74
- tqdm 4.35.0

For developers:

- pytest 5.1.2 (for tests only)

2.2 Installation Guide

Clone or download the source code of the project and run

```
# pip install /path/to/DeepNOG
```

If you are a developer, run

```
# pip install -e /path/to/DeepNOG
```

instead.

Congratulation, now you should have the `deepnog` command installed!

3 User's Guide

3.1 Usage Examples

DeepNOG can be used through calling the above installed `deepnog` command with a protein data file.

Its basic syntax looks as follows: **deepnog** *proteins.file* [*options*]

Common example usages are:

- `deepnog proteins.faa`
  - Predicts the orthologous groups of proteins in *proteins.faa* and writes them into *out.csv* created in the current directory.
- `deepnog proteins.faa --out prediction.csv`

[1]DeepNOG is tested with the package versions listed but can work with deviating version numbers.

  - Predicts the orthologous groups of proteins in *proteins.faa*. Instead of writing into *out.csv* creates *prediction.csv* as an output file.

- `deepnog proteins.faa.gz --tab`
  - Predicts the orthologous groups of proteins in the gzipped *proteins.faa.gz* file. Instead of semicolon separated, generate tab separated output-file called *out.csv*.

3.2 Basic Commands

-h, --help	Show help message explaining command line options and usage.
--version	Show program version.
-o <*file*>, **--out** <*file*>	Path where to store the output-file containing the orthologous group predictions. Defaults to *out.csv*
-ff <*format*>, **--fformat** <*format*>	File format of protein sequences. Default value is *fasta*. Must be supported by Biopythons Bio.SeqIO class. It is not necessary to indicate if a file is gzipped or not.
--verbose *N*	Define verbosity of DeepNOGs output written to stdout or stderr. 0 only writes errors to stderr which cause DeepNOG to abort and exit. 1 also writes warnings to stderr if e.g. a protein without an ID was found and skipped. 2 additionally writes general progress messages to stdout. 3 (default) includes a dynamic progress bar of the prediction stage using tqdm.
-d {*auto,cpu,gpu*}, **--device** {*auto,cpu,gpu*}	Define device for calculating protein sequence classification. Default is *auto* which chooses a GPU if available, otherwise CPU is chosen.
--tab	If set, output will be tab-separated instead of ;-separated.

3.3 Advanced Commands

3.3.1 Options for GPU Usage

The following commands are intended only to be changed, if calculating on a GPU.

-nw *N*, **--num-workers** *N*	Number of subprocesses (workers) to use for data loading. Set to a value <= 0 to use single-process data loading. Per default, *N* is set to 0. Caution: Multi-process data loading on a CPU can be more inefficient than single-process data loading!
-bs *N*, **--batch-size** *N*	Batch size used for prediction. Defines how many sequences should be forwarded in the network at once. Defaults to a batch size of one, where the protein sequences are sequentially classified by the neural network without the possibility of leveraging parallelism (optimal for CPU). Higher batch-sizes than the default one can speed up the prediction significantly if on a GPU. But If on a CPU, they can be slower as smaller ones due to the increased average sequence length in the convolution step due to zero-padding every sequence in each batch.

3.3.2 Options to Customize Used Neural Network

4 Developer's Guide

To register a new neural network in DeepNOG, two things have to be provided:

- A .py file defining the custom neural network class (using PyTorch).
- A .pt or .pth file storing the trained parameters (weights) of the network.

-a <*architecture*>, **--architecture** <*architecture*>	Neural network architecture to use for classification. Currently, only *deepencoding* is supported (default setting). The chosen architecture has to match the name of a neural network class in the models folder (see developer guide in Section 4).
-w <*file*>, **--weights** <*file*>	Optional, define a path to a custom neural networks weights file (.pt or .pth) to use for classification.

-db <*database*>, **--database** <*database*>	Define database to classify against. Currently, only the default option *eggNOG5* is supported.
-t *N*, **--tax** *N*	`Optional.` Taxonomic level to use in specified database. Currently, levels *1* and *2* (indicating root and bacterial level in eggNOG 5.0, respectively), are supported.

To extend DeepNOG, it is essential to understand its folder structure as explained in Section 4.1. How a custom neural network class has to be designed is explained in Section 4.2. What has to be stored in the trained parameters (weights) file, is explained in Section 4.3.

4.1 Folder Structure

The folder structure of DeepNOG looks as follows:

```
/
├─ deepnog
│   ├─ models
│   │   └─ deepencoding.py
│   ├─ tests
│   ├─ dataset.py
│   └─ deepnog.py
└─ setup.py
```

The `models` folder holds the python class definition files of all supported neural networks. Each neural network has its own class definition file. A custom neural network model can be chosen by its filename given to *deepnog* via the *--architecture* command.

```
/
└─ ${DEEPNOG_DATA}
    └─ parameters
        └─ eggNOG5
            ├─ 1
            │   └─ deepencoding.pth
            └─ 2
                └─ deepencoding.pth
```

Trained weights are stored in the `parameters` folder with the deepnog data directory (default ~/deepnog_data, or set individually with DEEPNOG_DATA environment variable). Each supported database should have its own subfolder in the `parameters` folder. The database to use can be told to *deepnog* by specifying the corresponding folder name using the *--database* command. Each database folder has to be structure into taxonomic levels. Which taxonomic level to use has to be specified via the *--tax* command. trained weights files are stored under their respective database and taxonomic level. DeepNOG expects the weights file to have the same name as the corresponding architecture defined in `models`. Alternatively, DeepNOG can use a custom weights file by setting the path to the weights file using the *--weights* commands.

The `tests` folder holds software tests written in pytest for *deepencoding.py*, *dataset.py* and *deepnog.py*. If any changes are made to any of these files, it is highly recommended to run pytest from the `deepnog` directory.

The *dataset.py* file defines the dataset classes and helper functions for usage with neural network models written in PyTorch. To add custom neural networks to DeepNOG, it should not be necessary to change this file. Details on the dataset classes can be found in the code documentation.

The *deepnog.py* file holds the main function and defines the high-level workflow of DeepNOG. Again, it should not be necessary to make any changes to this file if one plans to extend DeepNOG with other neural networks. However, details about *deepnog.py* can be found in the code documentation.

The *setup.py* file as well as the many *__init__.py* files omitted in the listed directory tree for readability purposes, are used to package DeepNOG and should not be changed if one wants to add a new neural network.

4.2 Custom Neural Network Class

A custom neural network class can be added to DeepNOG by placing the class definition file into the `models` folder. It is very important that the name of the defining .py file as well as the class name coincide so that it can be correctly loaded by *deepnog.py*. In the example of DeepEncoding, the neural network class is called *deepencoding* and defined in the file *deepencoding.py*.

The custom neural network class needs to define a constructor which receives two arguments:

1. The first argument corresponds to a dictionary containing all key-value pairs stored in the parameters file corresponding to the custom neural network class. It should be used to correctly initialize the neural network architecture (e.g. defining kernel sizes or number of output units in the last layer). It should not be used to initialize the parameter values of the neural network. DeepNOG does handle loading the *state_dict* from your parameters file and setting all weight values to the trained one's for you automatically.
2. The calculation device which will either be set to *cpu* or *gpu*.

Furthermore, it needs to define a classic *forward* function with one argument corresponding to a batch of sequences. The sequences are a PyTorch tensor of shape (batch_size, sequence_len). The amino acids in the protein sequences are already translated to numerals. For the translation, the map outlined in table 1. Zero is reserved to zero-pad the sequence.

A	C	D	E	F	G	H	I	K	L	M	N	P
1	2	3	4	5	6	7	8	9	10	11	12	13
Q	R	S	T	V	W	Y	B	X	Z	J	U	O
14	15	16	17	18	19	20	21	22	23	24	25	26

Table 1: Translation used for the amino acids in the protein sequences.

As an output the *forward* function should return a PyTorch tensor of shape (batch_size, n_classes). For each sequence in the batch, it should store the prediction confidence that it is associated to sequence being a member of class 0, class 1, class 2, ..., class n_classes-1.

4.3 Structure of the Parameters File

The parameters file is a .ph or .pth file created by saving the parameters of a trained model using *torch.save*. It is very important to note that the parameters file is not only used to store the concrete weight values but also to store hyperparameters about the model. This allows for a more generic definition and broader application of the custom neural network classes. For example, in the case of pretraining the same neural network on two different databases could result in two differently sized output layers (different number of classes). Then, it is not necessary to have two distinct neural network class files. One can use the same neural network class definition file and besides storing the trained weights, also store the number of output units in the parameter file. This number will be accessible in the constructor of the neural network class through its first argument as explained in the previous Section.

More concretely, the parameters file is created by saving a dictionary object using *torch.save*. How the dictionary has to be structured, is explained in table 2. Then, the parameters file has to be placed either in the `parameters` folder or specified via the *--weights* command as outlined in Section 4.1.

	Name (key)	**Description**
Necessary	model_state_dict	Store the state_dict of the model which corresponds to all the (trained) weights used. (See PyTorch tutorials for what is the state_dict.) DeepNOG automatically handles loading and setting the trained weights for the model based on this entry.
Optional	User specified	Hyperparameters about the neural network architecture to be used in the constructor of the custom neural network class.
Optional	threshold	Set a prediction confidence threshold (float between 0 and 1). If set, DeepNOG will automatically extract the set threshold and treat all protein sequences with prediction below this threshold as not associated to any class included in the model (see Section 1.2.3 for details).

Table 2: Elements of the dictionary to create a correct parameters file.

References

Jaime Huerta-Cepas et al. eggnog 5.0: a hierarchical, functionally and phylogenetically annotated orthology resource based on 5090 organisms and 2502 viruses. *Nucleic Acids Res.*, 47(Database issue):D309–D314, 2019.