Retraining DEBInitNet

As the size of the Add-my-Pet (AmP) collection grows, more data is available to train the machine learning models. More data to train will increase the performance of the machine learning model, which will improve the accuracy of the parameter predictions. More accurate initial parameter values will reduce calibration time and execution errors.

This document explains the necessary steps to retrain the DEBInitNet model. Retraining should occur when the training set size increases by 300 species. Note that the size of the training set is not equal to the size of the AmP collection. Only species that have the necessary input zero-variate datasets and are of the considered typified models are included.

1. Requirements

To retrain the DEBInitNet, both MATLAB and Python need to be installed. MATLAB should have a more recent version than R2018b. Python should be version 3.12.

For MATLAB, ensure that the latest version of DEBtool is installed. This is best in order to assess if there any change in DEBtool functions caused inadvertently an error or incompatibility with the DEBInitNet code. In terms of toolboxes, the DEBtool implementation of DEBInitNet uses the Symbolic Math Toolbox to convert units (the version requirement for MATLAB is due to this). The Parallel Computing Toolbox is also useful, as it drastically reduces the computational time required to evaluate the performance of a retrained model.

For Python, create an environment and install the packages in the file `requirements.txt`. This file describes all the necessary packages needed to run the Python code.

2. Updating the dataset

2.1. Download the AmP collection

To build the dataset for training DEBInitNet, the AmP collection must be first downloaded. This can be achieved by running the python script `download_amp_files.py`:

```
python -m project_code.data.download_amp_files
```

With the default settings, the script will download the latest version of all files in the AmP collection, including new species, if they are more recent than the ones in disk. To force download the files in AmP, set the argument `overwrite` to `True`.

The species should be downloaded to the folder `data\species`. To download to another folder change the path in the file `filepaths.csv`.

2.2. Re-estimation of species parameters (optional)

To obtain the dataset to train the DEBInitNet model, the species in AmP were re-estimated starting from the parameters in the AmP until convergence. To repeat this, the MATLAB script `project_code\data\parallel_run_all_estimations.m` can be used. The script will find the list of species in `data\species` and re-estimate the parameters until convergence. To choose a different species list of species, edit the variable `speciesList` in the beginning of the script.

The script will output a .csv file with initial and final loss and parameters as well as execution times and errors that occurred. This file can be analyzed to validate the re-estimation.

Warning: this is both a labor-intensive and time-consuming step. It is likely that for some species execution errors will occur, which may need to be solved. Although the script runs in parallel, which means it runs faster on CPUs with more cores, it will probably take several days to complete.

The species in the repository have already been re-estimated. The difference in parameters values will be small for the majority of species, and the estimation procedures will improve over time, meaning that this step is not particularly worthwhile. It is described here for completeness.

2.3. Building the dataset

To build the dataset, run the MATLAB script `project_code\data\build_dataset_from_AmP.m`. The dataset will be saved in the folder `data\raw` with the name `raw_amp_dataset.csv`. For reproducibility, a copy of the dataset is also named with the current date.

The next step is to process the dataset in order to obtain the train, validation, and test splits. To do this, run the Jupyter notebook in folder `notebooks` named "`Data Preprocessing.ipynb`". The notebook will generate the datasets automatically and save them in the folder `data\processed`. The dataset for the DEBInitNet model will be in the folder named `data\processed\init_net`, with files for the full dataset and for each data split.

The test set will contain the same species even as the collection grows, in order to have a basis for comparison between versions of the models.

3. Model Training

3.1. Retraining

To retrain the DEBInitNet, simply run the training Python script with the new version of the dataset:

```
python -m project_code.train.train_pytorch_model
```

The hyperparameters of the neural network are already set in the script. After the preprocessing above (section 2.3), the new version of the dataset should still have the same name, meaning that no change is needed to the training script.

The model will be saved in the folder `results\init_net\models`. The name of the file will be based on the performance in the validation sent and will have the format

`log_Q_{value}_MLP.pth`, where `value` is the log Q metric (logarithm of the accuracy ratio) for the model in the validation set.

The performance in the test set is also automatically performed, and will be saved with the same file name, but in .csv format in the folder `results\init_net\test_performance`.

3.2. Hyperparameter tuning (optional)

Hyperparameter tuning is not required to be done after each retraining. It should occur after a large increase in the dataset size, namely 1000 species or every third retraining. The hyperparameters should not change significantly with small size increases of the data, so it is unnecessary to do so more frequently.

To tune the hyperparameters, run the Python script calibrate_pytorch_model.py:

```
python -m project_code.train.calibrate_pytorch_model
```

The script will find the best hyperparameters based on the validation data. The model will be automatically retrained after tuning and saved in the folder `results\init_net\models`. The name of the model will be based on the performance in the validation sent and will have the format `log_Q_{value}_MLP.pth`, where `value` is the log Q metric (logarithm of the accuracy ratio) for the model in the validation set.

The performance in the test set will be saved with the same file name, but in .csv format in the folder `results\init_net\test_performance`.

4. Model performance evaluation

This section details the steps to evaluate the performance of the model, as was done for the first version of DEBInitNet and as described in the paper. These steps are time consuming, but are required to assess whether the model still performs its intended function well and hopefully better.

It is expected that if the prediction accuracy improves, both feasibility and initialization performance should also improve.

4.1. Parameter prediction accuracy

To evaluate the performance of the model at predicting species parameters, run the Jupyter notebook `notebooks\Evaluate ML models.ipynb`. The notebook will load the model, get the parameter predictions for all species in the dataset and plot those predictions versus the dataset values. The notebook will also display the performance metrics for the set.

The file `results\parameter_predictions\MLP_predictions.csv` will contain the parameter predictions for each species in the dataset. This file is used by the scripts below to assess feasibility and initialization performance.

4.2. Feasibility

To check how often the model produces feasible parameter sets, run the MATLAB script `project_code\evaluate\parallel_check_feasibility.m`. The MATLAB script will check if the parameters outputted by the DEBInitNet pass the filter for the typified model of the species.

The script runs in parallel and there is a time limit on running the filter to avoid cases where the differential equations get stuck. It should take only a few minutes to run.

The results are saved in a file named `results\feasibility\MLP_feasibility.csv` and they can be analyzed by running the Jupyter notebook `notebooks\Compare models on feasibility.ipynb`.

4.3. Initialization performance

To assess the initialization performance, the first step is to run the MATLAB script `project_code\evaluate\parallel_compute_deb_loss.m`. This script computes the loss function with the parameters provided by the DEBInitNet but only for species that had feasible predictions.

The results are saved in a file named:

`results\deb_model_loss\MLP_deb_model_loss.csv`.

The script runs in parallel and there is a time limit on computing the loss to avoid cases where the differential equations get stuck. It should take only a few minutes to run.

The next step is to run the parameter estimation from the parameters provided by the DEBInitNet by running the MATLAB script:

`project_code\evaluate\parallel_evaluate_initialization_method.m`

The script runs in parallel and there is a time limit of one hour for running the estimation. The estimation only occurs for species in the test set. It is likely that this script takes several hours to run.

The results are saved in a file named `results\initialization\MLP_initialization.csv` and can be analyzed by running the Jupyter notebook `notebooks\Compare models on initialization performance.ipynb`

5. Porting the model to DEBtool

5.1. Converting the model from Python to MATLAB

The conversion of the model from Python to MATLAB is done automatically by running the Jupyter notebook `notebooks\Convert models to MATLAB.ipynb`. The notebook will generate a .mat file named `project_code\debtool_port\DEBInitNet\deb_init_net.mat`. This file contains the parameters of the neural network and is loaded by the implementation in DEBtool to run the initialization method.

5.2. Updating the model version in DEBtool

The final result of the retraining is the .mat file mentioned above. This file contains both the newly trained weights and biases of the neural network, as well as the hyperparameters.

To update the new version of DEBInitNet to DEBtool, only the .mat file needs to be replaced.

5.3. Testing the DEBtool implementation

As a final step, the DEBtool implementation should be tested for execution errors. This can be accomplished with the MATLB script:

`project_code\debtool_port\tests\test_init_DEBInitNet.m`

A copy of this script is also present in the DEBtool package with the following path:

`DEBtool_M\lib\pet\DEBInitNet\tests\test_init_DEBInitNet.m`

For each species, the script will do 500 steps of Nelder-Mead starting from the parameters provided by the DEBInitNet method. The script runs in parallel for all species in the AmP collection and will take a few hours to run.

The results are saved in a file named `test_init_DEBInitNet_{today}.csv`, where `{today}` is the date when the script was run. The results can be analyzed by running the Jupyter notebook `notebooks/DEBtool Port Test Analysis.ipynb`.

If after running the script, the section “Errors in DEBInitNet code” is empty, then no errors in the DEBInitNet code were found.