StructInfer

Release 0.1

StructInfer Developers

Sep 26, 2023

GETTING STARTED WITH STRUCTINFER!

CHAPTER

ONE

CONTENTS

1.1 About the Dataset

All datasets are provided in standard binary file format (NPY). The format stores all of the shape and data type information necessary to reconstruct the array correctly even on another machine with a different architecture.

1.1.1 Downloading NPY datasets

All datasets are hosted on https://structinfer.github.io/download/, where the links to download raw and split datasets in NPY format can be found at. After downloading the datasets, please move the corresponding files into /src/simulations/[name of the underlying graph]/directed [or undirected]/springs [or netsims]/. For the graph types are gene coexpression networks and landscape networks, the trajectories should be saved under /src/simulations/[name of the underlying graph]/undirected/springs [or netsims]/. For the others, please save them under /src/simulations/[name of the underlying graph]/directed/springs [or netsims]/.

1.1.2 Loading datasets in Python

After downloading an NPY dataset, it is easy to load it into Python with Numpy. You can also load a dataset from a directory of files in any supported structural data format by creating customized data-loading pipelines.

1.1.3 Naming policies of datasets

By default, daatasets contain all trajectories and underlying interacting graphs in each folder with certain training-validation-test split. For example, for the following properties: "directed", "CRNA", "15 nodes", "springs simulation", "noise-free", and "the first repetition" the data can be found at: /src/simulations/chemical_reaction_networks_in_atmosphere/directed/springs/. The files are:

- Trajectories for training: loc_train_springs15r1.npy, vel_train_springs15r1.npy,
- Groundtruth graphs for training: edges_train_springs15r1.npy,
- Trajectories for validation: loc_valid_springs15r1.npy, vel_valid_springs15r1.npy,
- Groundtruth graphs for validation: edges_valid_springs15r1.npy,
- Trajectories for test: loc_test_springs15r1.npy, vel_test_springs15r1.npy,
- Groundtruth graphs for test: edges_test_springs15r1.npy.

For the following properties: "directed", "BN", "30 nodes", "netsims simulation", "noise-free", and "the second repetition" the data can be found at: /src/simulations/brain_networks/directed/netsims/. The files are:

- Trajectories for training: bold_train_netsims30r2.npy,
- Groundtruth graphs for training: edges_train_netsims30r2.npy,
- Trajectories for validation: bold_valid_netsims30r2.npy,
- Groundtruth graphs for validation: edges_valid_netsims30r2.npy,
- Trajectories for test: bold_test_netsims30r2.npy,
- Groundtruth graphs for test: edges_test_netsims30r2.npy.

For the following properties: "undirected", "LN", "50 nodes", "netsims simulation", "noise level 2", and "the third repetition" the data can be found at: /src/simulations/landscape_networks/undirected/netsims/. The files are:

- Trajectories for training: bold_train_netsims50r3_n2.npy,
- Groundtruth graphs for training: edges_train_netsims50r3_n2.npy,
- Trajectories for validation: bold_valid_netsims50r3_n2.npy,
- Groundtruth graphs for validation: edges_valid_netsims50r3_n2.npy,
- Trajectories for test: bold_test_netsims50r3_n2.npy,
- Groundtruth graphs for test: edges_test_netsims50r3_n2.npy.

1.1.4 More comments

All of the trajectories are in the shape of: [trajectories, nodes, features, timesteps]. For trajectories generated by springs simulations, for example, with a directed graph consisting of 15 nodes and with the first repetition, both "loc_train_springs15r1.npy", and "vel_train_springs15r1.npy" have the shape of [8000, 15, 2, 49]. Meanwhile, the ground truth graph has the shape: [nodes, nodes], which is an adjacency matrix, and if the element at row i, column j is one, it represents that there is an directed edge from node i to j. In order to get the full features, we have to concatenate both files on the feature dimension, and obtain new trajectories with the shape: [8000, 15, 4, 49].

But for the trajectories generated by netsims simulations, for example, with a directed graph consisting of 30 nodes and with the second repetition, "bold_train_netsims30r2.npy" has the shape of [8000, 30, 1, 49].

The trajectories for validation and test, each have 2000 trajectories, respectively.

1.2 About the Structural Inference Methods

The structural inference methods benchmarked with StructInf are collected from multiple discinplinaries such as biology and computer science. We follow the original implementation of these methods, but with slight modification to intergrating data loading and metric calculations. In the following paragraphs, the implementation of the structural inference methods in this work will be discussed in details.

1.2.1 Structural Inference Methods in this Work

Methods	Paper	Official Implementation	Our Implementation
ppcor: An R Package for a Fast Calculation to Semi-partial Correlation Coefficients (ppcor)	Link	Link	/src/models/ppcor
TIGRESS: Trustful Inference of Gene REgulation using Stability Selection (TIGRESS)	Link	Link	/src/models/TIGRESS
ARACNE: An Algorithm for the Reconstruction of Gene Regulatory Networks in a Mammalian Cellular Context (ARACNe)	Link	Link	/src/models/ARACNE
Large-Scale Mapping and Validation of Escherichia coli Transcriptional Regulation from a Compendium of Expression Profiles (CLR)	Link	Link	/src/models/CLR
Gene Regulatory Network Inference from Single-Cell Data Using Multivariate Information Measures (PIDC)	Link	Link	/src/models/PIDC/
Inferring Causal Gene Regulatory Networks from Coupled Single-Cell Expression Dynamics Using Scribe (Scribe)	Link	Link	/src/models/scribe
dynGENIE3: dynamical GENIE3 for the inference of gene networks from time series expression data (dynGENIE3)	Link	Link	/src/models/dynGENIE3
Inference of gene regulatory networks based on nonlinear ordinary differential equations (XGBGRN)	Link	Link	/src/models/GRNs_nonlinear_ODEs
Neural Relational Inference for Interacting Systems (NRI)	Link	Link	/src/models/NRI
Amortized Causal Discovery: Learning to Infer Causal Graphs from Time-Series Data (ACD)	Link	Link	/src/models/ACD
Neural Relational Inference with Efficient Message Passing Mechanisms (MPM)	Link	Link	/src/models/MPM
Iterative Structural Inference of Directed Graphs (iSIDG)	Link	Link	/src/models/iSIDG

1.2.2 Methods based on Classical Statistics

Unless otherwises specified, the following args are used to select the trajectories to be used for evaluation:

```
parser = add_option(parser, c("--data-path"), type="character", default="/work/projects/
↪bsimds/backup/src/simulations/",
                    help="The folder where data are stored.")
parser = add_option(parser, c("--save-folder"), type="character", default="",
                    help="The folder where resulting adjacency matrixes are stored.")
parser = add_option(parser, c("--b-portion"), type="numeric", default=1.0,
                    help="Portion of data to be used in benchmarking.")
parser = add_option(parser, c("--b-time-steps"), type="integer", default=49L,
                    help="Portion of time series in data to be used in benchmarking")
parser = add_option(parser, c("--b-network-type"), type="character", default="",
                    help="What is the network type of the graph.")
parser = add_option(parser, c("--b-directed"), action="store_true", default=FALSE,
                    help="Default choose trajectories from undirected graphs.")
parser = add_option(parser, c("--b-simulation-type"), type="character", default="",
                    help="Either springs or netsims.")
parser = add_option(parser, c("--b-suffix"), type="character", default="",
                    help='The rest to locate the exact trajectories. E.g. "50r1_n1" for␣
↪50 nodes, rep 1 and noise level 1. Or "50r1" for 50 nodes, rep 1 and noise free.')
```

ppcor

We use the official implementation of ppcor from the R package with a customized wrapper. Our wrapper will parse multiple arguments to select a set of targeted trajectories for inference, transform trajectories into a suitable format, feed each trajectory into the ppcor algorithm, and store the output into designated directories. Our implementation can be found at /src/models/ppcor in the provided Anonymous GitHub repository. The method is implemented in R with the help of NumPy Python package to store generated trajectories, reticulate from https://github.com/rstudio/reticulate to load Python variables into the R environment, stringr from https://stringr.tidyverse.org for string operation, and optparse from https://github.com/trevorld/r-optparse to produce Python-style argument parser.

TIGRESS

We use the official implementation of TIGRESS by the author at https://github.com/jpvert/tigress with a customized wrapper. Our wrapper will parse multiple arguments to select a set of targeted trajectories for inference, transform trajectories into a suitable format, feed each trajectory into the TIGRESS algorithm, and store the output into designated directories. Our implementation can be found at /src/models/TIGRESS in the provided Anonymous GitHub repository. The method is implemented in R with the help of NumPy Python package to store generated trajectories, reticulate from https://github.com/rstudio/reticulate to load Python variables into the R environment, stringr from https://stringr.tidyverse.org for string operation, and optparse from https://github.com/trevorld/r-optparse to produce Python-style argument parser.

1.2.3 Methods based on Information Theory

Unless otherwises specified, the following args are used to select the trajectories to be used for evaluation:

```
parser = add_option(parser, c("--data-path"), type="character", default="/work/projects/
↪bsimds/backup/src/simulations/",
                    help="The folder where data are stored.")
parser = add_option(parser, c("--save-folder"), type="character", default="",
                    help="The folder where resulting adjacency matrixes are stored.")
parser = add_option(parser, c("--b-portion"), type="numeric", default=1.0,
                    help="Portion of data to be used in benchmarking.")
parser = add_option(parser, c("--b-time-steps"), type="integer", default=49L,
                    help="Portion of time series in data to be used in benchmarking")
parser = add_option(parser, c("--b-network-type"), type="character", default="",
                    help="What is the network type of the graph.")
parser = add_option(parser, c("--b-directed"), action="store_true", default=FALSE,
                    help="Default choose trajectories from undirected graphs.")
parser = add_option(parser, c("--b-simulation-type"), type="character", default="",
                    help="Either springs or netsims.")
parser = add_option(parser, c("--b-suffix"), type="character", default="",
                    help='The rest to locate the exact trajectories. E.g. "50r1_n1" for␣
↪50 nodes, rep 1 and noise level 1. Or "50r1" for 50 nodes, rep 1 and noise free.')
```

ARACNe

We use the implementation of ARACNe by the Bioconductor package minet with a customized wrapper. Our wrapper will parse multiple arguments to select a set of targeted trajectories for inference, transform trajectories into a suitable format, feed each trajectory into the ARACNe algorithm, and store the output into designated directories. Our implementation can be found at /src/models/ARACNE in the provided Anonymous GitHub repository. The method is implemented by minet in R with the help of NumPy Python package to store generated trajectories, reticulate from https://github.com/rstudio/reticulate to load Python variables into the R environment, stringr from https://stringr.tidyverse.org for string operation, and optparse from https://github.com/trevorld/r-optparse to produce Python-style argument parser.

CLR

We use the implementation of CLR by the Bioconductor package minet with a customized wrapper. Our wrapper will parse multiple arguments to select a set of targeted trajectories for inference, transform trajectories into a suitable format, feed each trajectory into the CLR algorithm, and store the output into designated directories. Our implementation can be found at /src/models/CLR in the provided Anonymous GitHub repository. The method is implemented by minet in R with the help of NumPy Python package to store generated trajectories, reticulate from https://github.com/rstudio/reticulate to load Python variables into the R environment, stringr from https://stringr.tidyverse.org for string operation, and optparse from https://github.com/trevorld/r-optparse to produce Python-style argument parser.

PIDC

Following args are used to select the trajectories to be used for evaluation:

```
s = ArgParseSettings()
@add_arg_table s begin
    "--data-path"
        help = "The folder where data are stored."
      arg_type = String
      default = "/work/projects/bsimds/backup/src/simulations/"
    "--save-folder"
        help = "The folder where resulting adjacency matrixes are stored."
        arg_type = String
        required = true
    "--b-portion"
      help = "Portion of data to be used in benchmarking."
      arg_type = Float64
      default = 1.0
    "--b-time-steps"
      help = "Portion of data to be used in benchmarking."
      arg_type = Int
      default = 49
    "--b-shuffle"
      help = "Shuffle the data for benchmarking?"
      action = :store_true
      default = false
    "--b-network-type"
        help = "What is the network type of the graph."
        arg_type = String
      default = ""
    "--b-directed"
```

(continues on next page)

(continued from previous page)

```
      help = "Default choose trajectories from undirected graphs."
      action = :store_true
    "--b-simulation-type"
      help = "Either springs or netsims."
      arg_type = String
      default = ""
    "--b-suffix"
        help = "The rest to locate the exact trajectories. E.g. \"50r1_n1\" for 50 nodes,
↪ rep 1 and noise level 1. Or \"50r1\" for 50 nodes, rep 1 and noise free."
        arg_type = String
      default = ""
end
```

We use the official implementation of PIDC by the author at https://github.com/Tchanders/NetworkInference.jl with a customized wrapper. Our wrapper will parse multiple arguments to select a set of targeted trajectories for inference, transform trajectories into a suitable format, feed each trajectory into the PIDC algorithm, and store the output into designated directories. Our implementation can be found at /src/models/PIDC in the provided Anonymous GitHub repository. The method is implemented in Julia with the help of NumPy Python package to store generated trajectories, ArgParse.jl from https://github.com/carlobaldassi/ArgParse.jl to parse command line arguments, CSV.jl from https://github.com/JuliaData/CSV.jl to save and load .csv files, DataFrames.jl from https://github.com/JuliaData/DataFrames.jl to manipulate data array, and NPZ.jl from https://github.com/fhs/NPZ.jl to load .npy into the Julia environment.

Scribe

Following args are used to select the trajectories to be used for evaluation:

```
parser.add_argument('--data-path', type=str,
                    default="/work/projects/bsimds/backup/src/simulations/",
                    help="The folder where data are stored.")
parser.add_argument('--save-folder', type=str, required=True,
                    help="The folder where resulting adjacency matrixes are stored.")
parser.add_argument('--b-portion', type=float, default=1.0,
                    help='Portion of data to be used in benchmarking.')
parser.add_argument('--b-time-steps', type=int, default=49,
                    help='Portion of time series in data to be used in benchmarking.')
parser.add_argument('--b-shuffle', action='store_true', default=False,
                    help='Shuffle the data for benchmarking?')
parser.add_argument('--b-network-type', type=str, default='',
                    help='What is the network type of the graph.')
parser.add_argument('--b-directed', action='store_true', default=False,
                    help='Default choose trajectories from undirected graphs.')
parser.add_argument('--b-simulation-type', type=str, default='',
                    help='Either springs or netsims.')
parser.add_argument('--b-suffix', type=str, default='',
                help='The rest to locate the exact trajectories. E.g. "50r1_n1" for 50␣
↪nodes, rep 1 and noise level 1. Or "50r1" for 50 nodes, rep 1 and noise free.')
parser.add_argument('--pct-cpu', type=float, default=1.0,
                    help='Percentage of number of CPUs to be used.')
```

We optimize the official implementation of Scribe by the author at https://github.com/aristoteleo/Scribe-py with a customized wrapper. Our wrapper will parse multiple arguments to select a set of targeted trajectories for inference, transform trajectories into a suitable format, feed each trajectory into the Scribe algorithm, and store the output into des-

ignated directories. Our implementation has customized causal_network.py and information_estimators.py scripts so as to modify the hyperparameters directly from command line arguments. We also have optimized the parallel support and computation efficiency and kept minimal functionality for benchmarking purposes, at the same time maintaining its general mechanism. Our implementation can be found at /src/models/scribe in the provided Anonymous GitHub repository. The method is implemented in Python with the help of NumPy package to store generated trajectories and tqdm from https://github.com/tqdm/tqdm to create progress bars.

1.2.4 Methods based on Tree Algorithms

Following args are used to select the trajectories to be used for evaluation:

```
parser.add_argument('--data-path', type=str,
                    default="/work/projects/bsimds/backup/src/simulations/",
                    help="The folder where data are stored.")
parser.add_argument('--save-folder', type=str, required=True,
                    help="The folder where resulting adjacency matrixes are stored.")
parser.add_argument('--b-portion', type=float, default=1.0,
                    help='Portion of data to be used in benchmarking.')
parser.add_argument('--b-time-steps', type=int, default=49,
                    help='Portion of time series in data to be used in benchmarking.')
parser.add_argument('--b-shuffle', action='store_true', default=False,
                    help='Shuffle the data for benchmarking?')
parser.add_argument('--b-network-type', type=str, default='',
                    help='What is the network type of the graph.')
parser.add_argument('--b-directed', action='store_true', default=False,
                    help='Default choose trajectories from undirected graphs.')
parser.add_argument('--b-simulation-type', type=str, default='',
                    help='Either springs or netsims.')
parser.add_argument('--b-suffix', type=str, default='',
                help='The rest to locate the exact trajectories. E.g. "50r1_n1" for 50␣
↪nodes, rep 1 and noise level 1. Or "50r1" for 50 nodes, rep 1 and noise free.')
parser.add_argument('--pct-cpu', type=float, default=1.0,
                    help='Percentage of number of CPUs to be used.')
```

dynGENIE3

We optimize the official Python implementation of dynGENIE3 by the author at https://github.com/vahuynh/dynGENIE3 with a customized wrapper. Our wrapper will parse multiple arguments to select a set of targeted trajectories for inference, transform trajectories into a suitable format, feed each trajectory into the dynGENIE3 algorithm, and store the output into designated directories. Following the principle of maintaining dynGENIE's general mechanism, we have modified the dynGENIE3.py script so as to tune the hyperparameters directly from command line arguments, increase computation efficiency on big datasets, enable calculation of self-influence, and retain minimal functionality for benchmarking purposes. Our implementation can be found at /src/models/dynGENIE3 in the provided Anonymous GitHub repository. The method is implemented in Python with the help of NumPy package to store generated trajectories.

XGBGRN

We use the official implementation of XGBGRN by the author at https://github.com/lab319/GRNs_nonlinear_ODEs with a customized wrapper. Our wrapper will parse multiple arguments to select a set of targeted trajectories for inference, transform trajectories into a suitable format, feed each trajectory into the XGBGRN algorithm, and store the output into designated directories. Our implementation can be found at /src/models/GRN nonlinear ODEs in the provided Anonymous GitHub repository. The method is implemented in Python with the help of NumPy package to store generated trajectories.

1.2.5 Methods based on VAEs

In general, we added following arguments to the argparse variable in these methods:

```
parser.add_argument('--save-probs', action='store_true', default=False,
                    help='Save the probs during test.')
parser.add_argument('--b-portion', type=float, default=1.0,
                    help='Portion of data to be used in benchmarking.')
parser.add_argument('--b-time-steps', type=int, default=49,
                    help='Portion of time series in data to be used in benchmarking.')
parser.add_argument('--b-shuffle', action='store_true', default=False,
                    help='Shuffle the data for benchmarking.')
parser.add_argument('--data-path', type=str, default='',
                    help='Where to load the data. May input the paths to edges_train of␣
↪the data.')
parser.add_argument('--b-network-type', type=str, default='',
                    help='What is the network type of the graph.')
parser.add_argument('--b-directed', action='store_true', default=False,
                    help='Default choose trajectories from undirected graphs.')
parser.add_argument('--b-simulation-type', type=str, default='',
                    help='Either springs or netsims.')
parser.add_argument('--b-suffix', type=str, default='',
    help='The rest to locate the exact trajectories. E.g. "50r1_n1" for 50 nodes, rep 1␣
↪and noise level 1.'
         ' Or "50r1" for 50 nodes, rep 1 and noise free.')
```

NRI

We use the official implementation code by the author from https://github.com/ethanfetaya/NRI with customized data loaders for our chosen datasets. The customized data loaders are named “load_customized_springs_data” and “load_customized_netsims_data”. Both of them are implemented in the “utils.py” file. The metric calculation pipeline is integrated into the “test” function. Besides that, the remaining part are in consistent with its official implementation. The code of our implementation can be found at /src/models/NRI in the provided Anonymous GitHub repository.

ACD

We use the official implementation code by the author https://github.com/loeweX/AmortizedCausalDiscovery with a customized data loader for our datasets. The customized data loader is named "load_data_customized", and is implemented in "data_loader.py". The metric calculation pipeline is integrated into the function "forward_pass_and_eval" of "foward_pass_and_eval.py" file. Besides that, the remaining part are in consistent with its official implementation. The code of our implementation can be found at /src/models/ACD in the provided Anonymous GitHub repository.

MPM

We use the official implementation code by the author at https://github.com/hilbert9221/NRI-MPM with a customized data loader for our chosen datasets. The customized data loader function is named "load_customized_data", and with data preprocessing functions "load_nri" and "load_netsims". The first function is implemented in "run.py", while the rest are implemented in "load.py". The metric calculation pipelines are integrated into the "test" function of "XNRIIns" class in "XNRI.py" file. Besides that, the remaining part are in consistent with its official implementation. The code of our implementation can be found at /src/models/MPM in the provided Anonymous GitHub repository.

iSIDG

We use the official implementation sent by the authors. We modified it with a customized data loader function: "load_data_benchmark", which is implemented in "utils.py". Besides that, the remaining part are in consistent with its official implementation. The code of our implementation can be found at /src/models/iSIDG in the provided Anonymous GitHub repository.

1.3 How to Reproduce the Results

Before the reproduction of the results in out benchmark, please follow the instructions in *About the Dataset* to download the datasets, and store them correctly under every subfolder. Then the structural inference methods in the benchmark and their results can be reproduced with following steps.

1.3.1 Methods based on Classical Statistics

ppcor

Requirements

To configure the environment, you can create a conda environment and install the *environment.yml* by:

```
$> conda env create -f environment.yml --name ppcor
```

Our environment included:

- r-base=4.1.3
- r-matrix=1.5_3
- r-optparse=1.7.3
- r-ppcor=1.1
- r-reticulate=1.28
- r-stringi=1.7.12

- r-stringr=1.5.0

Reproduction Examples

Reproduce the results of ppcor in the noise-free trajectories generated by NetSims simulation, and by Brain Networks with 15 nodes, with the first repetition number:

```
$> Rscript run.R --save-folder="./results" --b-network-type="brain_networks" --b-
↪directed --b-simulation-type="netsims" --b-suffix="test_netsims15r1.npy" &
```

Reproduce the results of ppcor in the noise-free trajectories generated by NetSims simulation, and by Brain Networks with 30 nodes, with the second repetition number:

```
$> Rscript run.R --save-folder="./results" --b-network-type="brain_networks" --b-
↪directed --b-simulation-type="netsims" --b-suffix="test_netsims30r2.npy" &
```

Reproduce the results of ppcor in the noisy trajectories generated by NetSims simulation, and by Brain Networks with 50 nodes, with the third repetition number, with two levels of added Gaussian noise:

```
$> Rscript run.R --save-folder="./results" --b-network-type="brain_networks" --b-
↪directed --b-simulation-type="netsims" --b-suffix="test_netsims50r3_n2.npy" &
```

Reproduce the results of ppcor in the noise-free trajectories generated by NetSims simulation, by Brain Networks with 30 nodes, with the second repetition number, and with 5 time steps:

```
$> Rscript run.R --save-folder="./results" --b-network-type="brain_networks" --b-
↪directed --b-simulation-type="netsims" --b-suffix="test_netsims30r2.npy" --b-time-
↪steps 5 &
```

TIGRESS

Requirements

To configure the environment, you can create a conda environment and install the *environment.yml* by:

```
$> conda env create -f environment.yml --name TIGRESS
```

Our environment included:

- r-base=4.1.3
- r-doparallel=1.0.17
- r-foreach=1.5.2
- r-matrix=1.5_4
- r-optparse=1.7.3
- r-reticulate=1.28
- r-stringr=1.5.0

Reproduction Examples

Reproduce the results of TIGRESS in the noise-free trajectories generated by NetSims simulation, and by Brain Networks with 15 nodes, with the first repetition number:

```
$> Rscript run.R --save-folder="./results" --b-network-type="brain_networks" --b-
↪directed --b-simulation-type="netsims" --b-suffix="test_netsims15r1.npy" &
```

Reproduce the results of TIGRESS in the noise-free trajectories generated by NetSims simulation, and by Brain Networks with 30 nodes, with the second repetition number:

```
$> Rscript run.R --save-folder="./results" --b-network-type="brain_networks" --b-
↪directed --b-simulation-type="netsims" --b-suffix="test_netsims30r2.npy" &
```

Reproduce the results of TIGRESS in the noisy trajectories generated by NetSims simulation, and by Brain Networks with 50 nodes, with the third repetition number, with two levels of added Gaussian noise:

```
$> Rscript run.R --save-folder="./results" --b-network-type="brain_networks" --b-
↪directed --b-simulation-type="netsims" --b-suffix="test_netsims50r3_n2.npy" &
```

Reproduce the results of TIGRESS in the noise-free trajectories generated by NetSims simulation, by Brain Networks with 30 nodes, with the second repetition number, and with 5 time steps:

```
$> Rscript run.R --save-folder="./results" --b-network-type="brain_networks" --b-
↪directed --b-simulation-type="netsims" --b-suffix="test_netsims30r2.npy" --b-time-
↪steps 5 &
```

1.3.2 Methods based on Information Theory

ARACNe

Requirements

To configure the environment, you can create a conda environment and install the *environment.yml* by:

```
$> conda env create -f environment.yml --name ARACNE
```

Our environment included:

- r-base=4.2.3
- r-biocmanager=1.30.20
- r-optparse=1.7.3
- r-reticulate=1.26
- r-stringi=1.7.12
- r-stringr=1.5.0
- bioconductor-minet

Reproduction Examples

Reproduce the results of ARACNe in the noise-free trajectories generated by NetSims simulation, and by Brain Networks with 15 nodes, with the first repetition number:

```
$> Rscript test.R --save-folder="./results" --b-network-type="brain_networks" --b-
↪directed --b-simulation-type="netsims" --b-suffix="test_netsims15r1.npy" &
```

Reproduce the results of ARACNe in the noise-free trajectories generated by NetSims simulation, and by Brain Networks with 30 nodes, with the second repetition number:

```
$> Rscript test.R --save-folder="./results" --b-network-type="brain_networks" --b-
↪directed --b-simulation-type="netsims" --b-suffix="test_netsims30r2.npy" &
```

Reproduce the results of ARACNe in the noisy trajectories generated by NetSims simulation, and by Brain Networks with 50 nodes, with the third repetition number, with two levels of added Gaussian noise:

```
$> Rscript test.R --save-folder="./results" --b-network-type="brain_networks" --b-
↪directed --b-simulation-type="netsims" --b-suffix="test_netsims50r3_n2.npy" &
```

Reproduce the results of ARACNe in the noise-free trajectories generated by NetSims simulation, by Brain Networks with 30 nodes, with the second repetition number, and with 5 time steps:

```
$> Rscript test.R --save-folder="./results" --b-network-type="brain_networks" --b-
↪directed --b-simulation-type="netsims" --b-suffix="test_netsims30r2.npy" --b-time-
↪steps=5 &
```

CLR

Requirements

To configure the environment, you can create a conda environment and install the *environment.yml* by:

```
$> conda env create -f environment.yml --name CLR
```

Our environment included:

- r-base=4.2.3
- r-biocmanager=1.30.20
- r-matrix=1.5_3
- r-optparse=1.7.3
- r-reticulate=1.26
- r-stringi=1.7.12
- r-stringr=1.5.0
- bioconductor-minet

Reproduction Examples

Reproduce the results of CLR in the noise-free trajectories generated by NetSims simulation, and by Brain Networks with 15 nodes, with the first repetition number:

```
$> Rscript test.R --save-folder="./results" --b-network-type="brain_networks" --b-
↪directed --b-simulation-type="netsims" --b-suffix="test_netsims15r1.npy" &
```

Reproduce the results of CLR in the noise-free trajectories generated by NetSims simulation, and by Brain Networks with 30 nodes, with the second repetition number:

```
$> Rscript test.R --save-folder="./results" --b-network-type="brain_networks" --b-
↪directed --b-simulation-type="netsims" --b-suffix="test_netsims30r2.npy" &
```

Reproduce the results of CLR in the noisy trajectories generated by NetSims simulation, and by Brain Networks with 50 nodes, with the third repetition number, with two levels of added Gaussian noise:

```
$> Rscript test.R --save-folder="./results" --b-network-type="brain_networks" --b-
↪directed --b-simulation-type="netsims" --b-suffix="test_netsims50r3_n2.npy" &
```

Reproduce the results of CLR in the noise-free trajectories generated by NetSims simulation, by Brain Networks with 30 nodes, with the second repetition number, and with 5 time steps:

```
$> Rscript test.R --save-folder="./results" --b-network-type="brain_networks" --b-
↪directed --b-simulation-type="netsims" --b-suffix="test_netsims30r2.npy" --b-time-
↪steps 5 &
```

PIDC

Requirements

To configure the environment, you have to install a Julia executable.

Our environment included:

- ArgParse
- CSV
- DataFrames
- NPZ
- NetworkInference

After installing Julia, you have to install packages in our project by:

1. Install it in Julia interactive session.

```
julia> using Pkg
julia> Pkg.instantiate()
```

2. Alternatively, install it in Julia REPL mode. On the shell:

```
$> julia --project=./PIDC/
```

On the Julia REPL mode:

```
(PIDC) pkg> instantiate
```

Reproduction Examples

Reproduce the results of PIDC in the noise-free trajectories generated by NetSims simulation, and by Brain Networks with 15 nodes, with the first repetition number:

```
$> julia --project=./PIDC/ -- run.jl --save-folder="./results" --b-network-type="brain_
↪networks" --b-directed --b-simulation-type="netsims" --b-suffix="test_netsims15r1.npy"␣
↪&
```

Reproduce the results of PIDC in the noise-free trajectories generated by NetSims simulation, and by Brain Networks with 30 nodes, with the second repetition number:

```
$> julia --project=./PIDC/ -- run.jl --save-folder="./results" --b-network-type="brain_
↪networks" --b-directed --b-simulation-type="netsims" --b-suffix="test_netsims30r2.npy"␣
↪&
```

Reproduce the results of PIDC in the noisy trajectories generated by NetSims simulation, and by Brain Networks with 50 nodes, with the third repetition number, with two levels of added Gaussian noise:

```
$> julia --project=./PIDC/ -- run.jl --save-folder="./results" --b-network-type="brain_
↪networks" --b-directed --b-simulation-type="netsims" --b-suffix="test_netsims50r3_n2.
↪npy" &
```

Reproduce the results of PIDC in the noise-free trajectories generated by NetSims simulation, by Brain Networks with 30 nodes, with the second repetition number, and with 5 time steps:

```
$> julia --project=./PIDC/ -- run.jl --save-folder="./results" --b-network-type="brain_
↪networks" --b-directed --b-simulation-type="netsims" --b-suffix="test_netsims30r2.npy"␣
↪--b-time-steps=5 &
```

Scribe

Requirements

To configure the environment, you can create a conda environment and install the *environment.yml* by:

```
$> conda env create -f environment.yml --name scribe
```

Our environment included:

- numpy=1.23.5
- pandas=1.5.2
- python=3.9.16
- pip: - scikit-learn==1.2.1 - scipy==1.10.0 - tqdm==4.64.1

Reproduction Examples

Reproduce the results of scribe in the noise-free trajectories generated by NetSims simulation, and by Brain Networks with 15 nodes, with the first repetition number:

```
$> python run.py --save-folder="./results" --b-network-type="brain_networks" --b-
↪directed --b-simulation-type="netsims" --b-suffix="test_netsims15r1.npy" &
```

Reproduce the results of scribe in the noise-free trajectories generated by NetSims simulation, and by Brain Networks with 30 nodes, with the second repetition number:

```
$> python run.py --save-folder="./results" --b-network-type="brain_networks" --b-
↪directed --b-simulation-type="netsims" --b-suffix="test_netsims30r2.npy" &
```

Reproduce the results of scribe in the noisy trajectories generated by NetSims simulation, and by Brain Networks with 50 nodes, with the third repetition number, with two levels of added Gaussian noise:

```
$> python run.py --save-folder="./results" --b-network-type="brain_networks" --b-
↪directed --b-simulation-type="netsims" --b-suffix="test_netsims50r3_n2.npy" &
```

Reproduce the results of scribe in the noise-free trajectories generated by NetSims simulation, by Brain Networks with 30 nodes, with the second repetition number, and with 5 time steps:

```
$> python run.py --save-folder="./results" --b-network-type="brain_networks" --b-
↪directed --b-simulation-type="netsims" --b-suffix="test_netsims30r2.npy" --b-time-
↪steps=5 &
```

1.3.3 Methods based on Tree Algorithms

dynGENIE3

Requirements

To configure the environment, you can create a conda environment and install the *environment.yml* by:

```
$> conda env create -f environment.yml --name dynGENIE3
```

Our environment included:

Our environment included:

- numpy=1.23.5
- pandas=1.5.2
- python=3.10.9
- scikit-learn=1.2.0
- scipy=1.10.0

Reproduction Examples

Reproduce the results of dynGENIE3 in the noise-free trajectories generated by NetSims simulation, and by Brain Networks with 15 nodes, with the first repetition number:

```
$> python run.py --save-folder="./results" --b-network-type="brain_networks" --b-
↪directed --b-simulation-type="netsims" --b-suffix="test_netsims15r1.npy" &
```

Reproduce the results of dynGENIE3 in the noise-free trajectories generated by NetSims simulation, and by Brain Networks with 30 nodes, with the second repetition number:

```
$> python run.py --save-folder="./results" --b-network-type="brain_networks" --b-
↪directed --b-simulation-type="netsims" --b-suffix="test_netsims30r2.npy" &
```

Reproduce the results of dynGENIE3 in the noisy trajectories generated by NetSims simulation, and by Brain Networks with 50 nodes, with the third repetition number, with two levels of added Gaussian noise:

```
$> python run.py --save-folder="./results" --b-network-type="brain_networks" --b-
↪directed --b-simulation-type="netsims" --b-suffix="test_netsims50r3_n2.npy" &
```

Reproduce the results of dynGENIE3 in the noise-free trajectories generated by NetSims simulation, by Brain Networks with 30 nodes, with the second repetition number, and with 5 time steps:

```
$> python run.py --save-folder="./results" --b-network-type="brain_networks" --b-
↪directed --b-simulation-type="netsims" --b-suffix="test_netsims30r2.npy" --b-time-
↪steps=5 &
```

XGBGRN

Requirements

To configure the environment, you can create a conda environment and install the *environment.yml* by:

```
$> conda env create -f environment.yml --name GRNs_nonlinear_ODEs
```

Our environment included:

Our environment included:

- numpy=1.23.5
- pandas=1.5.2
- py-xgboost-cpu=1.7.3
- python=3.10.9
- scikit-learn=1.2.0

Reproduction Examples

Reproduce the results of XGBGRN in the noise-free trajectories generated by NetSims simulation, and by Brain Networks with 15 nodes, with the first repetition number:

```
$> python run.py --save-folder="./results" --b-network-type="brain_networks" --b-
↪directed --b-simulation-type="netsims" --b-suffix="test_netsims15r1.npy" &
```

Reproduce the results of XGBGRN in the noise-free trajectories generated by NetSims simulation, and by Brain Networks with 30 nodes, with the second repetition number:

```
$> python run.py --save-folder="./results" --b-network-type="brain_networks" --b-
↪directed --b-simulation-type="netsims" --b-suffix="test_netsims30r2.npy" &
```

Reproduce the results of XGBGRN in the noisy trajectories generated by NetSims simulation, and by Brain Networks with 50 nodes, with the third repetition number, with two levels of added Gaussian noise:

```
$> python run.py --save-folder="./results" --b-network-type="brain_networks" --b-
↪directed --b-simulation-type="netsims" --b-suffix="test_netsims50r3_n2.npy" &
```

Reproduce the results of XGBGRN in the noise-free trajectories generated by NetSims simulation, by Brain Networks with 30 nodes, with the second repetition number, and with 5 time steps:

```
$> python run.py --save-folder="./results" --b-network-type="brain_networks" --b-
↪directed --b-simulation-type="netsims" --b-suffix="test_netsims30r2.npy" --b-time-
↪steps=5 &
```

1.3.4 Methods based on VAEs

NRI

Please install the required packages first.

Requirements

- Python >= 3.8
- Numpy >= 1.23.4
- pandas >= 1.5.1
- matplotlib >= 3.6.2
- sklearn >= 0.0.post1
- torch >= 1.13.1
- torchinfo >= 1.7.2
- tqdm >= 4.64.1

Arguments

- b-network-type: name of the graph type (in full name)
- b-directed: if called, will load data from directed graphs
- b-simulation-type: springs or netsims
- b-suffix: choose graph with node X, the Y repetition and with noise level K with format "XrY_nK". If use noise-free, omit "_nK"

Reproduction Examples

Run NRI with "chemical reaction networks in atmosphere (CRNA)", "directed", "15 nodes", "springs simulation", "noise-free", and "the first repetition" :

```
$> cd /src/models/NRI/
$> python3 train.py --b-network-type 'chemical_reaction_networks_in_atmosphere' --b-
↪directed --b-simulation-type 'springs' --b-suffix '15r1'
```

Run NRI with "brain networks (BN)", "directed", "netsims simulation", "30 nodes", "noise-free", and "the second repetition":

```
$> cd /src/models/NRI/
$> python3 train.py --b-network-type 'brain_networks' --b-directed --b-simulation-type
↪'netsims' --b-suffix '30r2'
```

Run NRI with "landscape networks (LN)", "directed", "netsims simulation", "50 nodes", "the third repetition", and "noise level 2":

```
$> cd /src/models/NRI/
$> python3 train.py --b-network-type 'landscape_networks' --b-simulation-type 'netsims' -
↪-b-suffix '50r3_n2'
```

ACD

Please install the required packages first.

Requirements

- Python >= 3.8
- Numpy >= 1.23.4
- pandas >= 1.5.1
- scipy >= 1.9.3
- sklearn >= 0.0.post1
- torch >= 1.13.1
- torchinfo >= 1.7.2
- tqdm >= 4.64.1

Arguments

- b-network-type: name of the graph type (in full name)
- b-directed: if called, will load data from directed graphs
- b-simulation-type: springs or netsims
- b-suffix: choose graph with node X, the Y repetition and with noise level K with format "XrY_nK". If use noise-free, omit "_nK"

Reproduction Examples

Run ACD with "chemical reaction networks in atmosphere (CRNA)", "directed", "15 nodes", "springs simulation", "noise-free", and "the first repetition" :

```
$> cd /src/models/ACD/
$> python3 train.py --b-network-type 'chemical_reaction_networks_in_atmosphere' --b-
↪directed --b-simulation-type 'springs' --b-suffix '15r1'
```

Run ACD with "brain networks (BN)", "directed", "netsims simulation", "30 nodes", "noise-free", and "the second repetition":

```
$> cd /src/models/ACD/
$> python3 train.py --b-network-type 'brain_networks' --b-directed --b-simulation-type
↪'netsims' --b-suffix '30r2'
```

Run ACD with "landscape networks (LN)", "directed", "netsims simulation", "50 nodes", "the third repetition", and "noise level 2":

```
$> cd /src/models/ACD/
$> python3 train.py --b-network-type 'landscape_networks' --b-simulation-type 'netsims' -
↪-b-suffix '50r3_n2'
```

MPM

Please install the required packages first.

Requirements

- Python >= 3.8
- Numpy >= 1.23.4
- scipy >= 1.9.3
- sklearn >= 0.0.post1
- torch >= 1.13.1
- torch-geometric >= 2.2.0
- torchinfo >= 1.7.2
- tqdm >= 4.64.1

Arguments

- b-network-type: name of the graph type (in full name)
- b-directed: if called, will load data from directed graphs
- b-simulation-type: springs or netsims
- b-suffix: choose graph with node X, the Y repetition and with noise level K with format "XrY_nK". If use noise-free, omit "_nK"

Reproduction Examples

Run ACD with "chemical reaction networks in atmosphere (CRNA)", "directed", "15 nodes", "springs simulation", "noise-free", and "the first repetition" :

```
$> cd /src/models/MPM/
$> python3 run.py --b-network-type 'chemical_reaction_networks_in_atmosphere' --b-
↪directed --b-simulation-type 'springs' --b-suffix '15r1'
```

Run ACD with "brain networks (BN)", "directed", "netsims simulation", "30 nodes", "noise-free", and "the second repetition":

```
$> cd /src/models/MPM/
$> python3 run.py --b-network-type 'brain_networks' --b-directed --b-simulation-type
↪'netsims' --b-suffix '30r2'
```

Run ACD with "landscape networks (LN)", "directed", "netsims simulation", "50 nodes", "the third repetition", and "noise level 2":

```
$> cd /src/models/ACD/
$> python3 run.py --b-network-type 'landscape_networks' --b-simulation-type 'netsims' --
↪b-suffix '50r3_n2'
```

iSIDG

Please install the required packages first.

Requirements

- Python >= 3.8
- Numpy >= 1.23.4
- pandas >= 1.5.1
- matplotlib >= 3.6.2
- sklearn >= 0.0.post1
- torch >= 1.13.1
- torchinfo >= 1.7.2
- tqdm >= 4.64.1

Arguments

- b-network-type: name of the graph type (in full name)
- b-directed: if called, will load data from directed graphs
- b-simulation-type: springs or netsims
- b-suffix: choose graph with node X, the Y repetition and with noise level K with format "XrY_nK". If use noise-free, omit "_nK"

Reproduction Examples

Run NRI with "chemical reaction networks in atmosphere (CRNA)", "directed", "15 nodes", "springs simulation", "noise-free", and "the first repetition" :

```
$> cd /src/models/iSIDG/
$> python3 train.py --b-network-type 'chemical_reaction_networks_in_atmosphere' --b-
↪directed --b-simulation-type 'springs' --b-suffix '15r1'
```

Run NRI with "brain networks (BN)", "directed", "netsims simulation", "30 nodes", "noise-free", and "the second repetition":

```
$> cd /src/models/iSIDG/
$> python3 train.py --b-network-type 'brain_networks' --b-directed --b-simulation-type
↪'netsims' --b-suffix '30r2'
```

Run NRI with "landscape networks (LN)", "directed", "netsims simulation", "50 nodes", "the third repetition", and "noise level 2":

```
$> cd /src/models/iSIDG/
$> python3 train.py --b-network-type 'landscape_networks' --b-simulation-type 'netsims' -
↪-b-suffix '50r3_n2'
```