Module 3: Inferring a Chemical Graph with Frequency of Fringe-trees from a Trained ANN Using MILP

`mol-infer/2L-model`

March 2, 2021

Contents

1 Outline **1**

2 Terms and Notation **2**

3 The Program's Input and Output **3**

3.1 Program Input 3

3.2 Input Data Format 4

3.3 Program Output 5

3.4 Output Data Format 6

4 Invoking the Program and a Computational Example **6**

4.1 Executing the Program 6

1 Outline

This note explains how to use an implementation of a mixed-integer linear programming (MILP) formulation that can infer a vector of graph descriptors given a target value and the weights and bias values of a trained artificial neural network (ANN).

The MILP is implemented in Python, using the PuLP modeling module of the COIN-OR package [5, 6, 7, 8].

To begin with, we give a list of the files that accompany this note.

- Folder `source_code`
  A folder containing four Python scripts that implement an MILP formulation for inferring feature vectors of chemical graphs from a trained ANN, and files containing minimum and maximum values of each descriptor in the MILP formulation.

  - `ann_inverter.py`
    An implementation of an MILP formulation for the Inverse problem on ANNs [1].
  - `MILP_2L_fc.py`
    A Python script that contains functions to initialize the variables and prepare the constraints for an MILP formulation for inferring chemical graphs with a prescribed topological structure [2].
  - `infer_graph_2L_fc.py`
    A Python script that prepares the data and executes the MILP formulation for given input data. Further details on the use of this script are given in Section 4.
  - `read_instance_2L_fc.py`
    A Python script that contains necessary functions to read the topological specification from a given textual file.
  - Folder `ANN`
    A folder containing information on trained artificial neural networks (ANNs) for six target properties: Boiling point (Bp), Melting point (Mp), Octanol/Water Partition Coefficient (Kow), Flash point (Fp), Solubility (Sl), and Lipophilicity (Lp). For each of the above six properties, `property` $\in \{$`BP`, `MP`, `KOW`, `FP`, `SL`, `LP`$\}$ five files are provided:

    * `property_desc.csv`
      A comma-separated value file containing descriptors.
    * `property_desc_norm.csv`
      A comma-separated value file containing normalized form of descriptors used in the training of the ANN.
    * `property_fringe.txt`
      A textual file containing all isomeric fringe trees in every dataset.
    * `property_biases.txt`
      A file containing the values of the biases of a trained ANN.
    * `property_weights.txt`
      A file containing the values of the weights of a trained ANN.

For each of the files, the data format is explained in Section 3, and an actual example is given in Section 4.

- Folder `topological_description`
  A folder containing six textual files each giving a chemical specification as detailed in [2].
  - `instance_a.txt`
  - `instance_b1.txt`
  - `instance_b2.txt`
  - `instance_b3.txt`
  - `instance_b4.txt`
  - `instance_c.txt`
  - `instance_d.txt`
- Folder `fringe_set`
  A folder containing textual files give the prepared fringe tree set.
  - `ins_a_fringe.txt`
  - `ins_c_fringe.txt`
  - `ins_d_fringe.txt`
    For each of the six properties, `property` $\in \{$`BP`, `MP`, `KOW`, `FP`, `SL`, `LP`$\}$ four files are provided:
  - `property_fr25.txt`
  - `property_fr30.txt`
  - `property_fr35.txt`
  - `property_fr40.txt`

The remaining of this note is organized as follows. Section 2 gives an explanation of the used terms and notation. Section 3 explains the input and output data of the program, and Section 4 gives a concrete example of input data and the results form the computation.

2 Terms and Notation

This section explains the terms and notation used in this note.

- **Feature vector**
  A *feature vector* stores numerical values of certain parameters, called *descriptors*. In this work, we choose graph-theoretical descriptors, such as number of non-hydrogen atoms, number of vertices of certain degree, etc.
- **Artificial neural network - ANN**
  Artificial neural networks are one of the methods in machine learning. They provide a means to construct a correlation function between pairs of feature vectors as input and target data as output.

- **Input, hidden, and output layer**
  We deal with the multilayer perceptron model of feed-forward neural networks. These neural networks are constructed of several *layers*. First comes the *input layer*, where each neuron takes as input one value of the feature vector. Next come the *hidden layers*, where the values from the input layer are propagated in a feed-forward manner, such that each node in one layer is connected to all the nodes of the next layer. Finally, the output is delivered at the *output layer*. We deal with predicting the value of a single target, and hence we assume that the output layer comprises a single node.

- **Weights**
  Each edge connecting two nodes in an ANN is assigned a real value, called a *weight*. Part of the *learning* process of ANNs is to determine values for each of the weights based on known pairs of feature vectors and target values.

- **Biases**
  Each node of an ANN except for the nodes in the input layer is assigned a real value, called a *bias*, which, just like the edge weights, is determined through the learning process.

- **Activation function**
  In an ANN, each node produces an output as a function, called the *activation function*, of its input. We assume that each node has the Rectified Linear-Unit (ReLU) function as its activation function, which can be expressed exactly in the MILP formulation for the inverse problem on ANNs [1].

- **Mixed-Integer Linear Programming (MILP)**
  A type of a mathematical programming problem where all the constraints are given as linear expressions, and some of the decision variables are required to take only integer values. For more details, see any standard reference, e. g. [3].

- **Graph**
  An abstract combinatorial construction comprising a finite set of *vertices*, and a finite set of *edges*, where each edge is a pair of vertices. We treat *undirected* graphs, i. e., graphs where edges are unordered pairs of vertices. For more information, see e. g. [4].

3 The Program's Input and Output

This section explains the format of the input and the output of the program. Section 3.1 illustrates an example of the program's input format, and Section 3.2 gives a concrete computational example. Following, Section 3.3 illustrates an example of the program's output format, and Section 3.4 gives a concrete computational example.

3.1 Program Input

This section gives an explanation of the input to the program.

First the input requires five textual files containing

- the descriptors and its normalized value in csv format

- the weights and biases of a trained ANN in textual format

- the fringe trees in textual format.

For a common prefix `TT` which the program accepts as a command-line parameter, these files must be saved with file names `TT_desc.csv`, `TT_desc_norm.csv`, `TT_weights.txt`, `TT_biases.txt` and `TT_fringe` for the files containing the descriptors, the normalized value of descriptors, the weights, the biases of a trained ANN, and fringe trees in dataset respectively. Next, comes the target value for which we wish to infer a chemical graph based on the trained ANN given above. Following is a chemical specification given in a textual file, as described in [2], as well as a filename prefix for the output files, which are described in Section 3.4

Finally, comes a choice of MILP solver program to be used. We can choose

- 1: CPLEX, a commercial MILP solver [9].

(Note, in this case the parameter `CPLEX_PATH` in the file `infer_graph_2L_fc.py` must be set to the correct path of the CPLEX program executable file.)

- 2: CBC, a free and open-source MILP solver. It comes together with the PuLP package for Python [5].

3.2 Input Data Format

This section presents an actual example of an input instance of the program. In particular, we give a concrete example of the three input files mentioned in Section 3.1.

The purpose of this program is to calculate a feature vector that will produce a desired output from a given trained ANN. Figure 1 gives an example of a trained ANN.

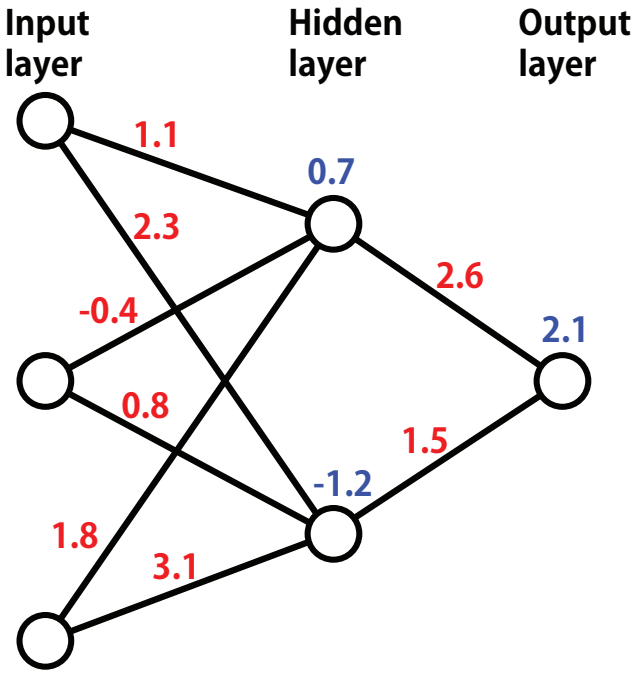

Figure 1: An example of a trained ANN. The ANN's weights are given in red numbers, and its biases in blue.

The information on the trained ANN is written in two text files, containing the information on

the ANN's weights and biases, respectively. First, we give the structure of the file that contains the information on the ANN's weights. The first line of this text file contains the information on the ANN's architecture, i. e., the number of nodes in each of its layers. From the second row and onward follows the information on the weights in the ANN. Each row contains the weights of the edges that are incident to one node of the ANN, first of the nodes in the input layer, and then for the nodes of the hidden layers. Following is a textual example for the ANN given in Fig. 1.

Organization of the text file containing weight data

```
3 2 1
1.1 2.3
-0.4 0.8
1.8 3.1
2.6
1.5
```

Next, comes the text file with the information on the ANN's biases. The bias values from Fig. 1 are given below.

Bias values

```
0.7
-1.2
2.1
```

Then comes text file containing data on the feature vector and the normalized feature vectors. The first line of files contains the names of the descriptors used in the feature vectors. Following form the second row onward, are the numerical values of the descriptors for each chemical graph in the training dataset, one row per chemical graph. For an example, please check the files `TT_desc.csv` and `TT_desc_norm.csv`, in the folder `ANN`, where $\texttt{TT} \in \{\texttt{BP}, \texttt{MP}, \texttt{KOW}, \texttt{FP}, \texttt{SL}, \texttt{LP}\}$.

3.3 Program Output

This section gives an explanation of the output of the program. If there exists a chemical graph with a feature vector that would result with the given target value as a prediction of the given trained ANN, the program will output the feature vector. In case such a chemical graph does not exist, the program will report this. The next section gives an explanation of the output of the program.

3.4 Output Data Format

This section describes the output data of the program as obtained on a personal computer.

Once invoked, the program will print some messages to the standard error stream, which appear on the terminal. Once the MILP solver completes the computation, the status of the computation is printed on the terminal.

Text output on the terminal

```
Initializing Time: 0.892        # Written to stderr
Number of variables: 7663       # Number of all variables
- Integer : 7395                # Number of integer variables
- Binary : 6050                 # Number of binary variables
Number of constraints: 9162     # Number of constraints
Status: Feasible                # Solution status
Solving Time: 5.056             # Time taken by the MILP solver
MILP y*: 3.264                  # Calculated target value in the MILP
```

Finally, if there exists a feasible solution the program writes to disk two text files. Recall that as a part of the input the program requires a filename used for the output files. Assume that the supplied parameter is `filename`. Then, the resulting two text files are named

- `filename.sdf`
- `filename_partition.txt`.

The file `filename.sdf` contains information on the inferred chemical graph in the SDF (Structure Data File) format. For more information see the official documentation (in English)

`http://help.accelrysonline.com/ulm/onelab/1.0/content/ulm_pdfs/direct/reference/ctfileformats2016.pdf`

for detail.

The file `filename_partition.txt` contains a decomposition of the graph from the file `filename.sdf` into acyclic subgraphs, as described in [2].

4 Invoking the Program and a Computational Example

This section explains how to invoke the program and explains a concrete computational example. Following is an example of invoking the program `infer_graph_2L_fc.py`.

4.1 Executing the Program

First make sure that the terminal is correctly directed to the location of the `source_code` folder, as described in Section 1.

```
python infer_graph_2L_fc.py trained_ann_filename_prefix target_value
        chemical_specification fringe_tree_set output_file_name
```

As an example we use the target property `KOW`, that is, the files from the trained ANN with filename prefix `KOW`, target value 3.2, the file `instance_a.txt` for a chemical specification, the file `ins_a_fringe.txt` for a fringe tree set, and CPLEX [9] as an MILP solver (parameter value 1).

```
python infer_cyclic_graphs_2L_fc.py ANN/KOW 3.2
     topological_description/instance_a.txt
     fringe_set/ins_a_fringe.txt result 1
```

By executing the above command, the following text should appear on the terminal prompt.

Text output on the terminal

```
Initializing Time: 0.892
Number of variables: 7663
- Integer : 7395
- Binary : 6050
Number of constraints: 9162
Status: Feasible
Solving Time: 5.056
MILP y*: 3.264
```

The contents of the output files `result.sdf` and `result_partition.txt` are as follows.

File `result.sdf`

```
1
MILP_2L
fc_vector
 39 42  0  0  0  0  0  0  0  0999 V2000
    0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    0.0000    0.0000    0.0000 O   0  0  0  0  0  0  0  0  0  0  0  0
    0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    0.0000    0.0000    0.0000 N   0  0  0  0  0  0  0  0  0  0  0  0
    0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
```

```
   0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
   0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
   0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
   0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
   0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
   0.0000    0.0000    0.0000 N   0  0  0  0  0  0  0  0  0  0  0  0
   0.0000    0.0000    0.0000 O   0  0  0  0  0  0  0  0  0  0  0  0
   0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
   0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
   0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
   0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
   0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
   0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
   0.0000    0.0000    0.0000 O   0  0  0  0  0  0  0  0  0  0  0  0
   0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
   0.0000    0.0000    0.0000 N   0  0  0  0  0  0  0  0  0  0  0  0
   0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
   0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
   0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
   0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
   0.0000    0.0000    0.0000 O   0  0  0  0  0  0  0  0  0  0  0  0
   0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
   0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
   0.0000    0.0000    0.0000 N   0  0  0  0  0  0  0  0  0  0  0  0
   0.0000    0.0000    0.0000 O   0  0  0  0  0  0  0  0  0  0  0  0
   0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
   0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
1 21  1  0  0  0  0
1 22  1  0  0  0  0
1 37  1  0  0  0  0
2  3  1  0  0  0  0
2 20  2  0  0  0  0
2 27  1  0  0  0  0
3 21  1  0  0  0  0
4  5  1  0  0  0  0
4  6  1  0  0  0  0
4 13  1  0  0  0  0
4 17  1  0  0  0  0
7  8  2  0  0  0  0
7 10  1  0  0  0  0
7 12  1  0  0  0  0
9 10  1  0  0  0  0
```

```
  9 11  1  0  0  0  0
 11 12  1  0  0  0  0
 12 13  1  0  0  0  0
 13 16  1  0  0  0  0
 14 15  2  0  0  0  0
 14 16  1  0  0  0  0
 14 18  1  0  0  0  0
 16 17  2  0  0  0  0
 17 19  1  0  0  0  0
 18 20  1  0  0  0  0
 19 20  1  0  0  0  0
 21 22  1  0  0  0  0
 21 23  1  0  0  0  0
 23 24  1  0  0  0  0
 24 25  1  0  0  0  0
 24 26  2  0  0  0  0
 27 28  1  0  0  0  0
 27 31  1  0  0  0  0
 27 32  1  0  0  0  0
 28 29  1  0  0  0  0
 28 30  1  0  0  0  0
 32 33  1  0  0  0  0
 33 34  1  0  0  0  0
 34 35  1  0  0  0  0
 34 36  1  0  0  0  0
 37 38  1  0  0  0  0
 38 39  3  0  0  0  0
M  END
$$$$
```

File `result_partition.sdf`

```
13
10
0 1
1 2 3 4 5 6 7 8 9 10 11
11
0 0
```

```
1 2 3 4 5 6 7 8 9 10 11
12
0 0
1 2 3 4 5 6 7 8 9 10 11
13
0 0
1 2 3 4 5 6 7 8 9 10 11
14
1 3
1 2 3 4 5 6 7 8 9 10 11
16
0 0
1 2 3 4 5 6 7 8 9 10 11
17
0 1
1 2 3 4 5 6 7 8 9 10 11
18
0 1
1 2 3 4 5 6 7 8 9 10 11
19
0 0
1 2 3 4 5 6 7 8 9 10 11
20
0 1
1 2 3 4 5 6 7 8 9 10 11
21
0 2
1 2 3 4 5 6 7 8 9 10 11
22
0 4
1 2 3 4 5 6 7 8 9 10 11
23
0 2
1 2 3 4 5 6 7 8 9 10 11
16
10 7 12
1 3
1 2 3 4 5 6 7 8 9 10 11
10 9 11
0 3
1 2 3 4 5 6 7 8 9 10 11
11 12
```

```
0 0
1 2 3 4 5 6 7 8 9 10 11
12 13
0 0
1 2 3 4 5 6 7 8 9 10 11
13 4 17
0 1
1 2 3 4 5 6 7 8 9 10 11
13 16
0 0
1 2 3 4 5 6 7 8 9 10 11
14 16
0 0
1 2 3 4 5 6 7 8 9 10 11
14 18
0 0
1 2 3 4 5 6 7 8 9 10 11
16 17
0 0
1 2 3 4 5 6 7 8 9 10 11
17 19
0 0
1 2 3 4 5 6 7 8 9 10 11
18 20
0 0
1 2 3 4 5 6 7 8 9 10 11
19 20
0 0
1 2 3 4 5 6 7 8 9 10 11
20 2 3 21
4 6
1 2 3 4 5 6 7 8 9 10 11
21 1 22
3 5
1 2 3 4 5 6 7 8 9 10 11
21 22
0 2
1 2 3 4 5 6 7 8 9 10 11
21 23
0 0
1 2 3 4 5 6 7 8 9 10 11
```

References

[1] T. Akutsu and H. Nagamochi. A Mixed Integer Linear Programming Formulation to Artificial Neural Networks, in Proceedings of the 2019 2nd International Conference on Information Science and Systems, pp. 215–220, https://doi.org/10.1145/3322645.3322683.

[2] T. Akutsu and H. Nagamochi. A Novel Method for Inference of Chemical Compounds with Prescribed Topological Substructures Based on Integer Programming. Arxiv preprint, arXiv:2010.09203

[3] J. Matousek and B. Gärtner. Understanding and Using Linear Programming. Springer, 2007.

[4] M. S. Rahman. Basic Graph Theory. Springer, 2017.

[5] A Python Linear Programming API, `https://github.com/coin-or/pulp`.

[6] Optimization with PuLP, `http://coin-or.github.io/pulp/`.

[7] The Python Papers Monograph, `https://ojs.pythonpapers.org/index.php/tppm/article/view/111`.

[8] Optimization with PuLP, `https://pythonhosted.org/PuLP/`.

[9] IBM ILOG CPLEX Optimization Studio 12.8 User Manual. `https://www.ibm.com/support/knowledgecenter/SSSA5P_12.8.0/ilog.odms.studio.help/pdf/usrcplex.pdf`.