`graph_energy` Overview

You Chen, Torin Stetina, Andrew Wildman

May 3, 2018

Summary

The goal of this project is twofold:

1. Develop a software framework that predicts the energy distribution of a group of microstates from their topological parameters
2. Validate this framework on two sets of example data, specifically an ideal ice model and a lattice gas model.

Motivation

Approach

The central prediction capabilities of `graph_energy` are provided through a machine learning algorithm. Because the relationship between the graph connections and the energy is expected to be non-linear, (e.g. for a hydrogen bond graph, four edges on a node will likely be lower in energy than three or five) a simple, feed forward neural net has been chosen as the prediction algorithm. Neural networks have a large meta-parameter space, so an optimization of these meta-parameters will be built into the software itself.

Use Cases

The `graph_energy` package has four main use cases:

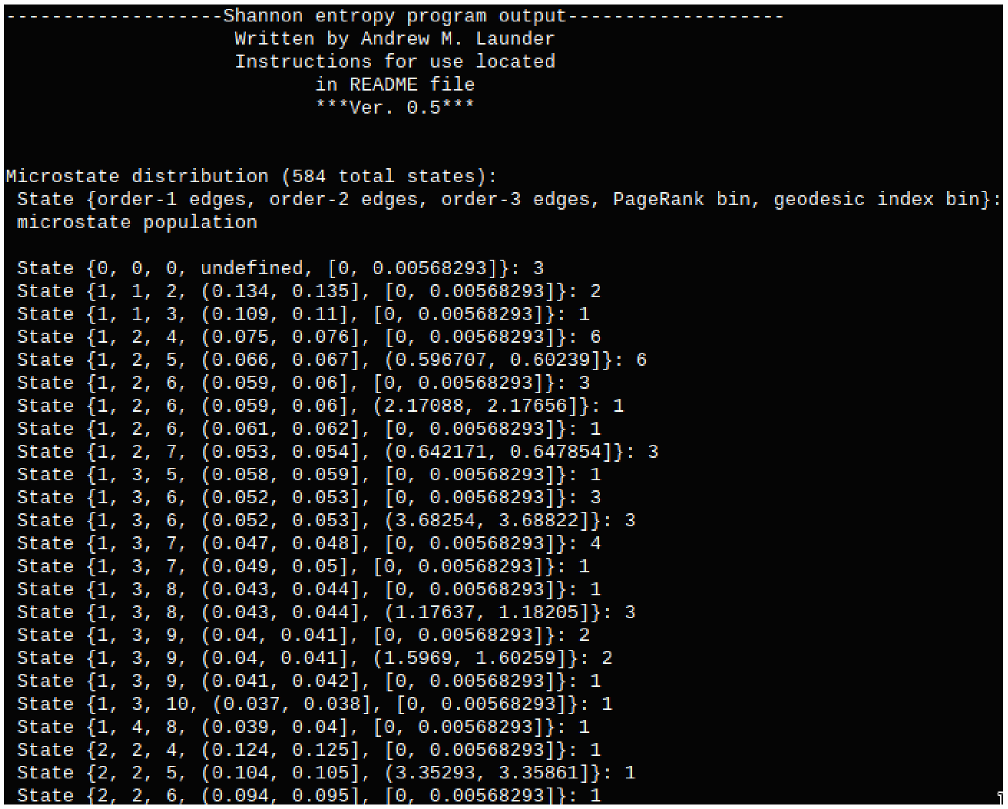
1. Train a neural net to reproduce the energy distribution of a set of microstates, given graph topological parameters that define the microstate, temperature, and population of the microstate at that temperature.
2. Evaluate the accuracy of nets trained on different sets of topological parameters at capturing the energy distribution.
3. Use previously trained neural nets to predict the energy distribution of microstates whose energy distribution was previously unknown.
4. Visualize the predicted energy distribution and, if available, contrast it with the known energy distribution.

Data Description

Note: This section is out of date once we get the new output that includes the true microstate energy.

The raw data comes from the Clark Group's Shannon entropy program. It consists of a set of `output.dat` files, each of which is generated at a given temperature and contain the topological parameter values of each microstate, as well as the population at that temperature. An example of an output.dat file is given in figure 1.

Software Design



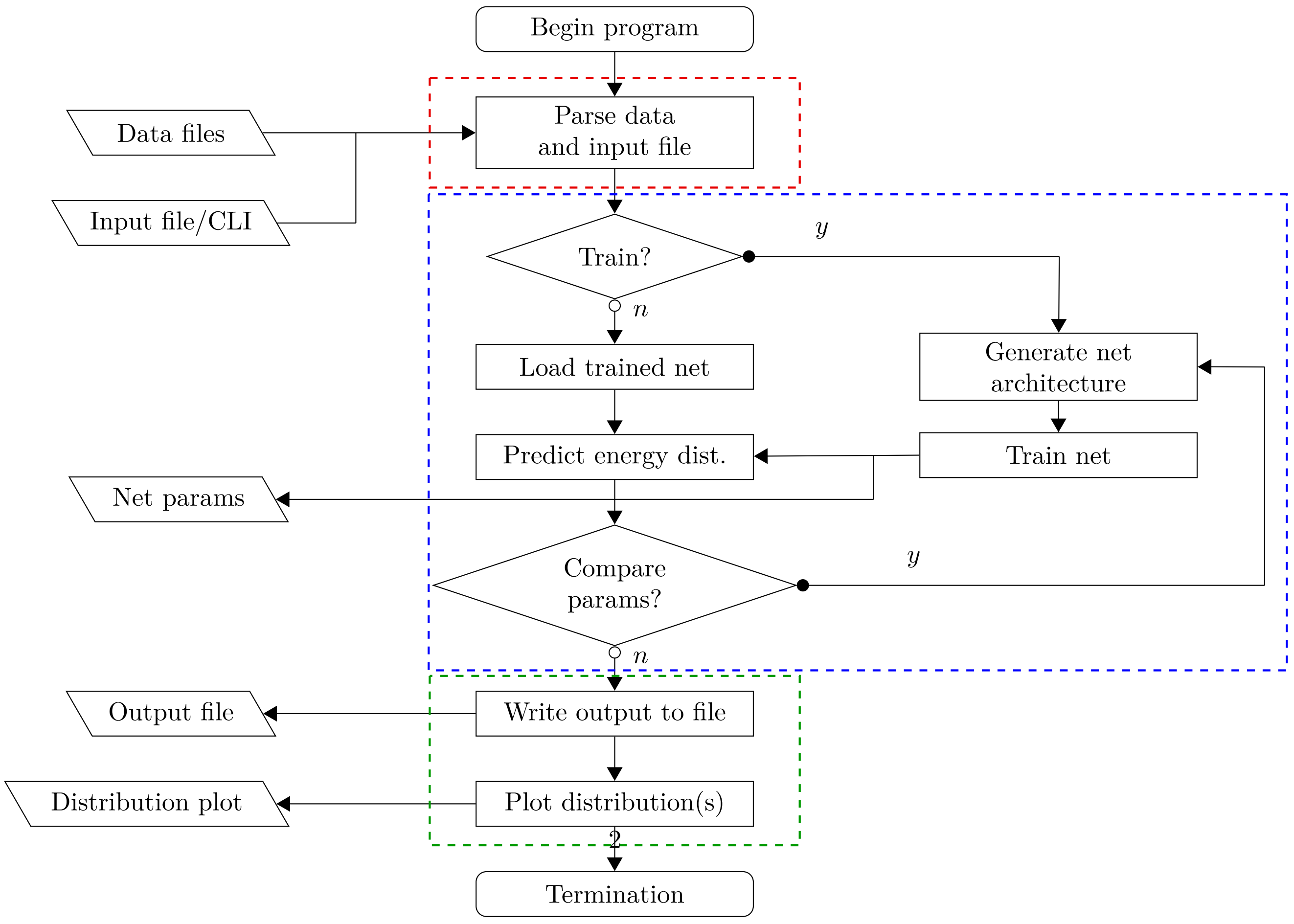
```
-------------------Shannon entropy program output-------------------
                    Written by Andrew M. Launder
                    Instructions for use located
                           in README file
                           ***Ver. 0.5***


Microstate distribution (584 total states):
 State {order-1 edges, order-2 edges, order-3 edges, PageRank bin, geodesic index bin}:
 microstate population

 State {0, 0, 0, undefined, [0, 0.00568293]}: 3
 State {1, 1, 2, (0.134, 0.135], [0, 0.00568293]}: 2
 State {1, 1, 3, (0.109, 0.11], [0, 0.00568293]}: 1
 State {1, 2, 4, (0.075, 0.076], [0, 0.00568293]}: 6
 State {1, 2, 5, (0.066, 0.067], (0.596707, 0.60239]}: 6
 State {1, 2, 6, (0.059, 0.06], [0, 0.00568293]}: 3
 State {1, 2, 6, (0.059, 0.06], (2.17088, 2.17656]}: 1
 State {1, 2, 6, (0.061, 0.062], [0, 0.00568293]}: 1
 State {1, 2, 7, (0.053, 0.054], (0.642171, 0.647854]}: 3
 State {1, 3, 5, (0.058, 0.059], [0, 0.00568293]}: 1
 State {1, 3, 6, (0.052, 0.053], [0, 0.00568293]}: 3
 State {1, 3, 6, (0.052, 0.053], (3.68254, 3.68822]}: 3
 State {1, 3, 7, (0.047, 0.048], [0, 0.00568293]}: 4
 State {1, 3, 7, (0.049, 0.05], [0, 0.00568293]}: 1
 State {1, 3, 8, (0.043, 0.044], [0, 0.00568293]}: 1
 State {1, 3, 8, (0.043, 0.044], (1.17637, 1.18205]}: 3
 State {1, 3, 9, (0.04, 0.041], [0, 0.00568293]}: 2
 State {1, 3, 9, (0.04, 0.041], (1.5969, 1.60259]}: 2
 State {1, 3, 9, (0.041, 0.042], [0, 0.00568293]}: 1
 State {1, 3, 10, (0.037, 0.038], [0, 0.00568293]}: 1
 State {1, 4, 8, (0.039, 0.04], [0, 0.00568293]}: 1
 State {2, 2, 4, (0.124, 0.125], [0, 0.00568293]}: 1
 State {2, 2, 5, (0.104, 0.105], (3.35293, 3.35861]}: 1
 State {2, 2, 6, (0.094, 0.095], [0, 0.00568293]}: 1
```

Figure 1: An example output file from the Clark Group's Shannon entropy program

Software flow



The rhombuses represent inputs or outputs to the program, rectangles represent processes, diamonds represent tests, and rounded boxes delimit the bounds of the program. The dashed boxes represent software components/classes, which will be described in the next section.

Component specification

Parser

The parser takes the `output.dat` files as well as the input file or command line parameters, then it converts them into the expected inputs for the parser.

Possible Inputs:Path to input file, OR Command line parameters that correspond to entries in the input file, OR Parameters passed to parser through a script Outputs: Exceptions:

Predictor

Possible Inputs:Outputs: Exceptions:

Visualizer

Possible Inputs:Outputs: Exceptions: