

# 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.

## 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

The raw data comes from the Clark Group's Shannon entropy program. It consists of a set of **parameters.dat** files, each of which is generated at a given temperature and contain the topological parameter values of each microstate, as well as the energy of each subgraph. The population of the microstates are given by the **output.dat** files.

An example of an **parameters.dat** file is given in figure 1.

## SOFTWARE DESIGN

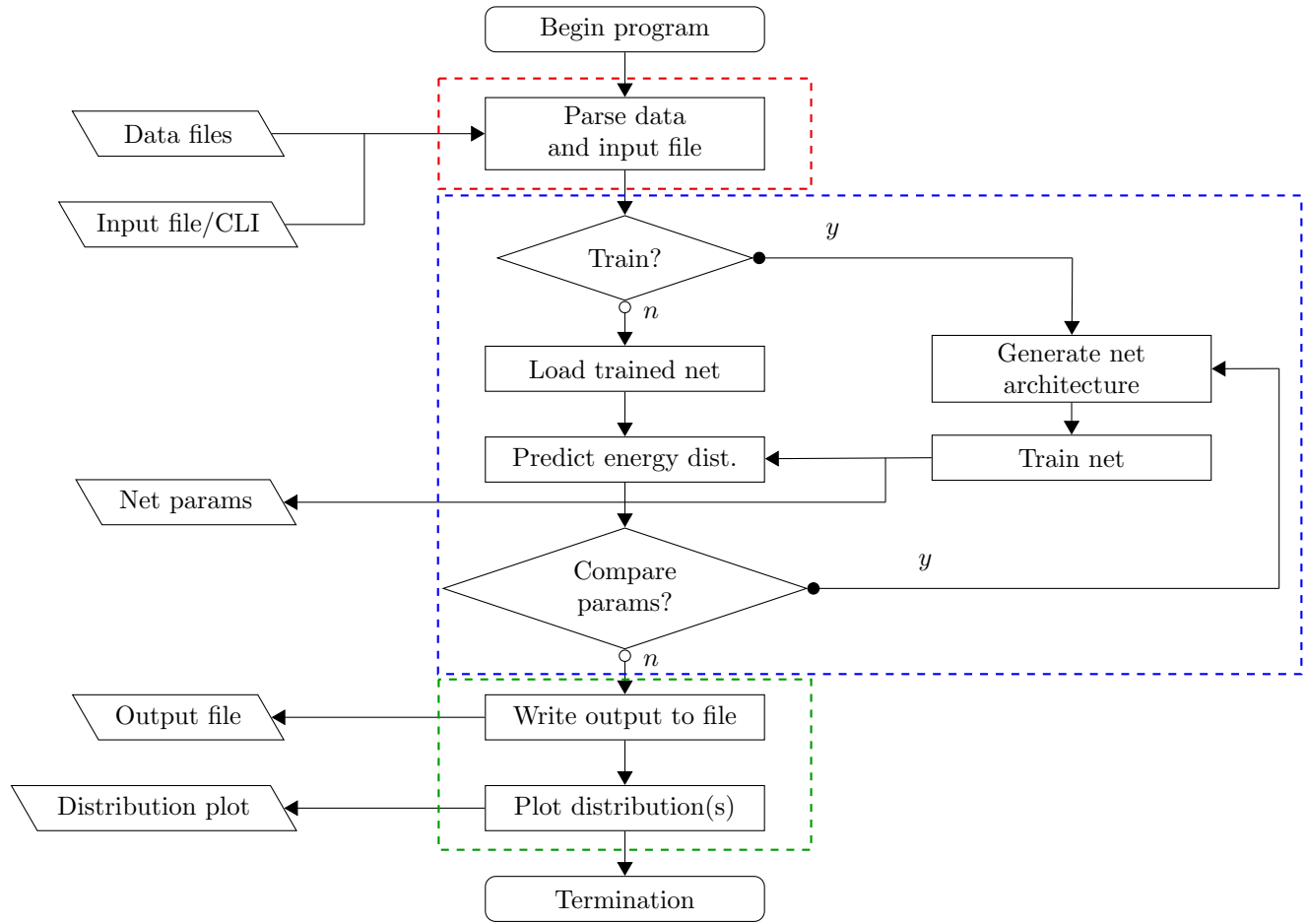
```

en3 pr3
677400 0.0448614
757200 0.0402949
755600 0.0394761
902600 0.0448785
841600 0.0476848
829600 0.0363251
579800 0.0641753
624200 0.0555526
751000 0.0439786
919800 0.0451364
932200 0.0448845
799200 0.0407025
883800 0.0461556
888600 0.0460787
667400 0.0465049
648200 0.0483965
946800 0.0407899
994600 0.0401097
996400 0.040645
687000 0.0442104
675000 0.0460045
738800 0.0418386
655800 0.0523525
879200 0.0468723
736000 0.0435292
735800 0.0451373
691600 0.0637105
485600 0.0474957
508600 0.0440968
806200 0.0380445
852600 0.047113
793400 0.0379986
945000 0.0408785
1.064e+06 0.0374703
1.028e+06 0.0380298
988200 0.0419273
755400 0.0401049
608200 0.049817
713200 0.0446963
683600 0.0443027
658400 0.0483679
847200 0.0488531
677200 0.0465364
916800 0.0454759
1.0022e+06 0.042442
912800 0.0449247
864400 0.0460682
814200 0.0363176
734200 0.0411699
711400 0.042819
856400 0.0462145
814200 0.0362791

```

Figure 1: An example output file from the Clark Group's Shannon entropy program. `en3` is the 3<sup>rd</sup> order subgraph energy, and `pr3` is the 3<sup>rd</sup> order page rank.

## 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 (red)

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

#### Input:

- `/path/to/parameters.dat`
- Job type: `train` or `predict`
- Meta parameter options for neural net

#### Outputs:

- Dataframe object from columns of `parameters.dat`
- Dictionary of meta parameters and job type for neural net

#### Exceptions:

- Handle wrong `.dat` file format

- Handle input file format errors

## **Predictor** (blue)

The predictor takes the metaparameter options and data from the parser, then does the following:

1. Checks type of job (predict vs train, which type of training)
2. Loads appropriate architecture for type of job
3. If training, trains the net
4. Predicts based on the trained net
5. Repeats steps 2-4 as many times as job requires

### Inputs:

- Dataframe whose rows are the subgraphs and whose columns are the topological parameters (and true energy)
- Dictionary of neural net meta-parameters

### Outputs:

- Dataframe whose rows are the subgraphs and whose columns are the topological parameters as well as the predicted (and true) energy
- (Optional) External file written with saved net weights and architecture

### Exceptions:

- Empty Dataframe
- Unsupported topological parameter
- (If train) No true energies
- (If predict) No matching pre-trained net for set of params

## **Visualizer** (green)

### Inputs:

- Dataframe with predicted energy (and true energy)
- Boolean for showing true energy

### Outputs:

- Plot of energy distribution
- Text file containing dataframe

### Exceptions

- Empty dataframe
- No predicted energy
- (If compare) No true energy