College of Science

Lecturer: Dr. Taheri

Spring 1402

Final project report

Graph mining course

Zahra Haghshenas

1. Introduction

The purpose of this work report is to provide details of a project aimed at predicting molecular properties using a Graph Neural Network (GNN) model. For this project, the FreeSolv dataset has been selected, which contains experimental and calculated hydration free energies of small molecules in water [10]. The dataset is a valuable resource for studying solvation properties and understanding the interactions between molecules and water.

The FreeSolv dataset includes a diverse set of small molecules, and each molecule is associated with its corresponding hydration free energy value. These values have been obtained through both experimental measurements and alchemical free energy calculations. The dataset is essential for developing and validating models that can accurately predict solvation free energies, which are crucial in various fields, including drug design, materials science, and environmental research.

In this report, we will provide a comprehensive description of the FreeSolv dataset, including its composition, data format, and statistical properties.

Additionally, we will outline the GNN model architecture used for predicting solvation free energies and describe the implementation details of the code employed for training and evaluating the model. The performance of the GNN model will be assessed using appropriate evaluation metrics, and the results will be analyzed to determine the effectiveness of the approach in predicting hydration free energies.

By leveraging the power of GNNs and utilizing the rich information provided by the FreeSolv dataset, we aim to develop a reliable and efficient model for predicting solvation free energies. The successful application of such models can significantly enhance our understanding of solvation phenomena and facilitate the design of molecules with improved solvation properties for various applications.

2 .Description of the dataset

The FreeSolv dataset is a database that contains experimental and calculated hydration free energies of small molecules in water. It was created to provide a benchmark dataset for the evaluation of solvation free energy prediction methods.

The dataset includes molecules with varying sizes and chemical properties. The hydration free energy represents the free energy change associated with the process of dissolving a solute molecule in water. It is an important property in understanding the solubility and behavior of molecules in aqueous environments.

The FreeSolv database contains both experimental and calculated values of hydration free energies. The experimental values are obtained through direct measurement techniques, such as isothermal titration calorimetry (ITC) or solute concentration measurements. The calculated values are obtained through alchemical free energy calculations, which involve computationally transforming the solute from the gas phase to the aqueous phase.

By combining experimental and calculated data, the FreeSolv database provides a comprehensive resource for evaluating the accuracy and reliability of solvation free energy prediction methods. It enables researchers to develop and validate computational models and algorithms for predicting hydration free energies, which have implications in drug discovery, chemical engineering, and other fields where solvation plays a crucial role.

3 . Code review

1. Package Installation:

   - The code begins with the installation of the `dgl` package using `!pip install dgl`. This package provides tools for deep learning on graph-structured data.

2. Importing Required Libraries:

- Various libraries are imported, including `os`, `dgl`, `numpy`, `networkx`, `torch`, `torch.nn`, `dgl.function`, `torch.nn.functional`, `shutil`, `torch.utils.data`, and `cloudpickle`. These libraries are essential for different functionalities used throughout the code.

3. Set Path and Directory Operations:

- This section involves setting the current directory, defining paths for saving models and data, creating necessary directories, and extracting the contents of a ZIP file using `shutil.unpack_archive()`.

4. Custom PyTorch Dataset Class:

- This section defines a custom dataset class called `DGLDatasetReg`. The class is designed for regression tasks using the Deep Graph Library (DGL). It provides methods for handling regression datasets and supports optional feature scaling.

5. Defining Train, Validation, and Test Sets:

- The code initializes train, validation, and test datasets using the `DGLDatasetReg` class. The `scaler` object is created from the train set to ensure consistent scaling for validation and test sets.

6. Analyzing Graph Properties:

- This section analyzes the graph properties of the concatenated dataset, such as the number of vertices, edges, and graphs. It also computes and prints the shape of the adjacency matrices for each graph.

7. Analyzing Graph Properties for a Single Graph:

- This part focuses on analyzing the properties of a single graph rather than all graphs in the dataset. It retrieves the first graph, calculates the number of vertices and edges, and computes the shape of the adjacency matrix.

8. Cumulative Counts of Graph Properties:

- This code calculates the cumulative counts of vertices and edges for all graphs in the dataset. It iterates over each graph and updates the cumulative counts. The shape of the adjacency matrix is also determined.

9. Accessing Global Features and Output Masks:

- These sections demonstrate accessing global features and output masks for the train, validation, and test sets. The code retrieves and prints the respective information for each set.

10. Checking for Empty Cells:

- This code checks if there are any empty cells (NaN values) in the train, validation, and test sets. It iterates over the samples and checks for NaN values in the labels tensor.

11. Accessing Global Features of Individual Molecules:

- This section demonstrates accessing and printing the global features of individual molecules from the train, validation, and test sets.

Review of Part 1:

The first part of your code contains several sections related to the setup and preparation of your project. Here's a breakdown of each section and its purpose:

3.1. Package Installation:

- The code begins with the installation of the `dgl` package using `!pip install dgl`. This package provides tools for deep learning on graph-structured data.

3.2. Importing Required Libraries:

- Various libraries are imported, including `os`, `dgl`, `numpy`, `networkx`, `torch`, `torch.nn`, `dgl.function`, `torch.nn.functional`, `shutil`, `torch.utils.data`, and `cloudpickle`. These libraries are essential for different functionalities used throughout the code.

3.3. Set Path and Directory Operations:

- This section involves setting the current directory, defining paths for saving models and data, creating necessary directories, and extracting the contents of a ZIP file using `shutil.unpack_archive()`.

3.4. Custom PyTorch Dataset Class:

- This section defines a custom dataset class called `DGLDatasetReg`. The class is designed for regression tasks using the Deep Graph Library (DGL). It provides methods for handling regression datasets and supports optional feature scaling.

3.5. Defining Train, Validation, and Test Sets:

- The code initializes train, validation, and test datasets using the `DGLDatasetReg` class. The `scaler` object is created from the train set to ensure consistent scaling for validation and test sets.

3.6. Analyzing Graph Properties:

- This section analyzes the graph properties of the concatenated dataset, such as the number of vertices, edges, and graphs. It also computes and prints the shape of the adjacency matrices for each graph.

3.7. Analyzing Graph Properties for a Single Graph:

- This part focuses on analyzing the properties of a single graph rather than all graphs in the dataset. It retrieves the first graph, calculates the number of vertices and edges, and computes the shape of the adjacency matrix.

3.8. Cumulative Counts of Graph Properties:

- This code calculates the cumulative counts of vertices and edges for all graphs in the dataset. It iterates over each graph and updates the cumulative counts. The shape of the adjacency matrix is also determined.

3.9. Accessing Global Features and Output Masks:

- These sections demonstrate accessing global features and output masks for the train, validation, and test sets. The code retrieves and prints the respective information for each set.

3.10. Checking for Empty Cells:

- This code checks if there are any empty cells (NaN values) in the train, validation, and test sets. It iterates over the samples and checks for NaN values in the labels tensor.

3.11. Accessing Global Features of Individual Molecules:

- This section demonstrates accessing and printing the global features of individual molecules from the train, validation, and test sets.

Overall, first part of our code focuses on data preparation, dataset creation, and analyzing the properties of the graphs in your dataset. It also includes sections for accessing global features and output masks.code focuses on data preparation, dataset creation, and analyzing the properties of the graphs in our dataset. It also includes sections for accessing global features and output masks.

The next part of our code focuses on training and evaluating a GNN model.

3.12. Collate Function:

- The `collate` function is defined to handle the batch data. It takes a list of tuples containing graphs, labels, masks, and globals and concatenates them along the batch dimension using `torch.stack()`. The function returns the concatenated batch data.

3.13. Loader Function:

- The `loader` function is defined to create data loaders for the training, validation, and test sets. It utilizes the `DataLoader` class from PyTorch and sets appropriate arguments such as batch size, collate function, drop last, shuffle, and number of workers. The function returns the data loaders for each set.

3.14. Training and Evaluation Loop:

- The code defines a `train_evaluate` function that encapsulates the training and evaluation loop for the GNN model. It initializes the model, optimizer, and other necessary variables. It then iterates over multiple epochs, performing the forward pass, loss calculation, backpropagation, and optimization steps for each batch in the training data. After each epoch, the model's performance is evaluated on the validation set using the `compute_score` function. The best model is saved based on the validation score, and the training progress is printed.

3.15. Training Epoch Function:

- The `train_epoch` function is called inside the `train_evaluate` function and handles the training loop for one epoch. It calculates the training loss for each batch, updates the model parameters, and returns the average training loss for the epoch.

3.16. Compute Score Function:

- The `compute_score` function evaluates the trained GNN model on a given dataset and calculates the root mean squared error (RMSE) score for the prediction tasks. It returns the RMSE score as a measure of the model's performance.

3.17. Loss Function:

- The `loss_func` function uses the `torch.nn.MSELoss` function from the `torch.nn.functional` module to calculate the mean squared error loss between the model's output and the label tensors.

3.18. Test Evaluation:

- The `test_evaluate` function loads the best model checkpoint, evaluates the model on the test dataset, and reports the test score using the `compute_score` function. It also prints the execution time.

3.19. Train and Test Execution:

- By calling the `train_evaluate` and `test_evaluate` functions one after the other, the code performs both training and testing of the GNN model. The `start_time` variable is used to calculate and print the total execution time for both operations.

our code focuses on training the GNN model using the defined functions and evaluating its performance on the validation and test sets. It utilizes data loaders, loss functions, and a training loop to optimize the model parameters.

we have defined different variations of the SAGEConv module and the GNN model for graph convolutional operations. Each variation uses different message and reduce functions to perform message passing and aggregation in the graph.

Here is a summary of the different variations we have defined:

3.20 SAGEConv1: This module uses element-wise addition as the message function and summation as the reduce function.

3.21. SAGEConv2: This module uses element-wise addition as the message function and summation as the reduce function, but it retrieves the aggregated neighbor features differently.

3.22. SAGEConv3: This module uses element-wise division as the message function and mean as the reduce function.

3.23. SAGEConv4: This module uses element-wise subtraction as the message function and mean as the reduce function.

3.24. SAGEConv5: This module uses element-wise multiplication as the message function and mean as the reduce function.

3.25. SAGEConv6: This module uses element-wise multiplication as the message function and summation as the reduce function.

3.26. SAGEConv7: This module uses element-wise multiplication as the message function and summation as the reduce function.

Each SAGEConv module is used in the GNN model to perform graph convolution operations. The GNN model consists of multiple SAGEConv layers, batch normalization layers, and ReLU activations. The forward method of the GNN model applies the SAGEConv layers in sequence and computes the mean of the resulting node features.

4 code output

After running the code on the FreeSolv dataset, obtained the following results:

- "Average valid score": This metric represents the average performance of the model on the validation set. It indicates how well the model generalizes to unseen data in predicting the solvation free energy of compounds.

- "Test score": This metric represents the performance of the model on the test set. It provides an estimation of the model's ability to predict the solvation free energy of compounds.

- "Execution time": This metric indicates the duration of code execution and data acquisition.

model	Average Valid Score:	Test Score:	Execution time:
GCN	2.027	2.004	23.207 seconds
Covsage: message copy and reduce mean	2.020	1.916	20.651 seconds
Covsage: message add and reduce sum	1.470	1.656	58.676 seconds
Covsage: message add and reduce sum	1.778	1.772	23.582 seconds
Covsage: message div and reduce max	inf	nan	5.581 seconds
Covsage: message sub and reduce mean	1.845	1.775	58.235 seconds
Covsage: message mul and reduce mean	1.960	1.857	53.092 seconds
Covsage: message mul and reduce sum	1.792	1.805	59.238 seconds
add layers 3 layers message mul and reduce sum	1.902	6.819	12.318 seconds

5.Result:

In conclusion, this report describes the work of a project focused on predicting the solvation free energy of compounds using a graph neural network model. The code utilizes the DGL and PyTorch libraries to process the FreeSolv dataset, train the model, and evaluate its performance. Based on the provided output criteria, the best-performing model is the Covsage: message add and reduce sum Because it has the lowest Test Score

These results demonstrate that the model achieves a reasonable level of performance in predicting the solvation free energy of compounds.