GNN to predict the Heat Capacity using QM9 Dataset.

Name: Rohan Joydhar
Email: rohan.joydhar@accenture.com

Introduction

Graph Neural Networks (GNNs) are a class of neural networks designed to perform tasks on graph-structured data. They leverage the inherent structure of graphs, capturing relationships between nodes and their features. This report discusses the implementation of a GNN, including dataset details, libraries and functions used,in the **notebook** to train a GNN model to predict Heat Capacity from smiles of molecules.

QM9 Dataset

The QM9 dataset is a widely used resource in the field of computational chemistry and machine learning. It consists of molecular data for 133,885 stable organic molecules, each containing up to 9 heavy atoms (C, O, N, F) and hydrogen atoms. The dataset was generated using density functional theory (DFT) calculations, specifically at the B3LYP/6-31G(2df,p) level of theory, which is a commonly used approximation method in quantum chemistry. Each molecule in the QM9 dataset is represented by a set of chemical and physical properties, such as:

1. **"smiles":** Containing smiles of molecule
2. **"mol_id":** Molecule ID (gdb9 index) mapping to the.sdf file
3. **"A":** Rotational constant (unit: GHz)
4. **"B":** Rotational constant (unit: GHz)
5. **"C":** Rotational constant (unit: GHz)
6. **"mu":** Dipole moment (unit: D)
7. **"alpha":** Isotropic polarizability (unit: Bohr^3)
8. **"homo":** Highest occupied molecular orbital energy (unit: Hartree)
9. **"lumo":** Lowest unoccupied molecular orbital energy (unit: Hartree)
10. **"gap":** Gap between HOMO and LUMO (unit: Hartree)
11. **"r2":** Electronic spatial extent (unit: Bohr^2)
12. **"zpve":** Zero point vibrational energy (unit: Hartree)
13. **"u0":** Internal energy at 0K (unit: Hartree)
14. **"u298":** Internal energy at 298.15K (unit: Hartree)
15. **"h298":** Enthalpy at 298.15K (unit: Hartree)
16. **"g298":** Free energy at 298.15K (unit: Hartree)
17. **"cv":** Heat capacity at 298.15K (unit: cal/(mol*K))
18. **"u0_atom":** Atomization energy at 0K (unit: kcal/mol)
19. **"u298_atom":** Atomization energy at 298.15K (unit: kcal/mol)
20. **"h298_atom":** Atomization enthalpy at 298.15K (unit: kcal/mol)
21. **"g298_atom":** Atomization free energy at 298.15K (unit: kcal/mol)

The QM9 dataset is valuable for several reasons:

Benchmark for Machine Learning Models: It serves as a benchmark dataset for developing and testing machine learning models aimed at predicting molecular properties. The relatively small size and simplicity of the molecules make it an ideal starting point for model development.

Exploration of Structure-Property Relationships: Researchers use the dataset to explore the relationships between molecular structure and properties, which can aid in the design of new materials and molecules with desired characteristics.

Data for Quantum Chemistry: The dataset provides a rich source of data for studying quantum chemical phenomena and validating theoretical models.

To train this GNN model a **subset** of QM9 dataset is used which contain **ten thousand data records** in it. This subset is further divided into **training**(8000 records) and **validation**(2000 records) sets.

Graph Neural Network (GNN)

A Graph Neural Network (GNN) is a type of neural network specifically designed to handle and process graph-structured data. Graphs are a versatile data structure consisting of nodes (vertices) connected by edges (links), and GNNs are capable of learning from this structure to perform various tasks.

Key Concepts of GNNs:

1. **Nodes and Edges**:
    - **Nodes** represent entities in the graph (e.g., people in a social network, atoms in a molecule).
    - **Edges** represent relationships or interactions between nodes (e.g., friendships, chemical bonds).
2. **Node Features and Edge Features**:
    - **Node Features**: Attributes associated with nodes (e.g., profile information, atomic number).
    - **Edge Features**: Attributes associated with edges (e.g., strength of interaction, bond type).
3. **Graph Convolutions**:
    - Similar to how convolutional layers in CNNs aggregate pixel information from local neighbourhoods in images, GNN layers aggregate information from neighbouring nodes to update the node's features.
    - The process of aggregating information from a node’s neighbours is often called "message passing."

4. **Adjacency Matrix**:
    - A matrix representing which nodes are connected by edges. It is used to guide the aggregation of node features in the graph convolution operation.
5. **Propagation and Aggregation**:
    - Nodes propagate their features to their neighbours.
    - Aggregation functions (e.g., sum, mean, max) combine the features from a node's neighbours to update its representation.

Advantages of GNNs:

- **Flexibility**: Can handle graphs of different sizes and shapes, making them suitable for a wide range of applications.
- **Expressiveness**: By learning from the structure and features of the graph, GNNs can capture complex patterns and relationships.
- **Applicability**: GNNs are used in diverse domains such as social networks, chemistry, biology, recommendation systems, and more.

Applications of GNNs:

1. **Social Network Analysis**: Predicting user behaviour, community detection, and friend recommendations.
2. **Chemoinformatics**: Predicting molecular properties, drug discovery, and chemical reaction outcomes.
3. **Recommendation Systems**: Personalizing content by modelling user-item interactions as graphs.
4. **Knowledge Graphs**: Enhancing information retrieval and answering complex queries.
5. **Traffic and Transportation**: Modeling and predicting traffic patterns and optimising routes.

Graph Neural Networks (GNNs) come in various types, each designed to handle different aspects of graph-structured data. Here are some of the main types of GNNs:

1. **Graph Convolutional Networks (GCNs)**:
    - **GCN**: The basic form of a GNN, which uses graph convolutional layers to aggregate feature information from a node's neighbours. It performs a weighted average of neighbour features, typically followed by a non-linear activation function.
    - **Spectral GCNs**: Based on spectral graph theory, these models use the graph Laplacian and its eigenvalues for convolution operations. The original GCN model proposed by Kipf and Welling falls into this category.
2. **Graph Attention Networks (GATs)**:
    - **GAT**: Introduces an attention mechanism to weigh the importance of neighbouring nodes differently. It allows the network to focus more on important nodes during the aggregation process.

3. **Graph Recurrent Neural Networks (Graph RNNs)**:
    - **Gated Graph Neural Networks (GGNNs)**: Extend recurrent neural networks (RNNs) to graphs. They use gated recurrent units (GRUs) to propagate information through the graph iteratively.
    - **Tree-LSTM**: A type of RNN designed to handle tree-structured data, a specific type of graph.
4. **Graph Autoencoders**:
    - **Variational Graph Autoencoders (VGAEs)**: Use encoder-decoder architectures to learn low-dimensional representations of nodes. VGAEs can be used for tasks like link prediction and graph generation.
5. **Message Passing Neural Networks (MPNNs)**:
    - **MPNNs**: A generalised framework that describes how nodes send and receive messages. It encompasses various GNN models, including GCNs and GATs.
6. **GraphSAGE (Graph Sample and Aggregation)**:
    - **GraphSAGE**: Generates node embeddings by sampling and aggregating features from a fixed-size neighbourhood. This method improves scalability to large graphs.
7. **Relational Graph Convolutional Networks (R-GCNs)**:
    - **R-GCNs**: Designed to handle graphs with multiple types of edges (relations). They are particularly useful for knowledge graphs where edges represent different relationships between entities.
8. **Dynamic Graph Neural Networks**:
    - **Dynamic GNNs**: Handle graphs that change over time, where nodes and edges can appear or disappear. Examples include Temporal Graph Networks (TGNs) and Recurrent GNNs for dynamic graphs.
9. **Graph Isomorphism Networks (GINs)**:
    - **GINs**: Aimed at addressing the expressiveness limitations of other GNNs, GINs are designed to be as powerful as the Weisfeiler-Lehman graph isomorphism test.
10. **Spatial GNNs**:
    - **Spatial GCNs**: These models perform convolutions directly on the graph structure in the spatial domain, rather than relying on spectral properties.

Applications:

- **Social Network Analysis**: GATs are often used for tasks like community detection and user recommendation due to their ability to focus on important connections.
- **Chemoinformatics**: GCNs and MPNNs are popular for predicting molecular properties because they can effectively capture the local structural information of molecules.
- **Knowledge Graphs**: R-GCNs are particularly suited for tasks involving multiple types of relationships, such as entity classification and link prediction.
- **Traffic Prediction**: Dynamic GNNs can be used to model and predict traffic patterns by capturing temporal changes in road networks.

Graph Convolutional Network (GCN)

A Graph Convolutional Network (GCN) is a type of neural network designed to work directly with graph-structured data. Unlike traditional neural networks, which operate on regular grid structures like images or sequences, GCNs can handle irregular structures where the data points (nodes) are connected by edges, forming a graph.

Key Concepts of GCN:

1. **Nodes and Edges**:
   - **Nodes** represent individual entities (e.g., atoms in a molecule).
   - **Edges** represent relationships or interactions between nodes (e.g., chemical bonds).
2. **Graph Convolution**:
   - In GCNs, convolutional operations are extended to graphs. Each node aggregates information from its neighbours to update its own representation.
   - The convolution operation in GCNs can be thought of as message passing, where each node receives and processes information from its neighbours.
3. **Adjacency Matrix**:
   - The adjacency matrix is a key component in GCNs, representing the connections between nodes. It is used to perform the convolution operation on the graph.
4. **Feature Matrix**:
   - Each node has associated features that describe its properties (e.g., atomic number for atoms). The feature matrix holds these node features.
5. **Layers**:
   - **Graph Convolutional Layers**: These layers perform the graph convolution operation, combining features from neighbouring nodes to create new node representations.
   - **Pooling Layers**: These layers aggregate node features to create a global graph representation, often used for tasks like graph classification or regression.
6. **Applications**:
   - GCNs are widely used in various domains, including social network analysis, recommender systems, and molecular property prediction.

Advantages of GCNs:

- **Capturing Local Structure**: GCNs effectively capture the local structure and relationships within the graph, which is crucial for understanding complex patterns.
- **Flexibility**: They can handle graphs of varying sizes and structures, making them suitable for a wide range of applications.
- **Expressiveness**: By aggregating information from neighbours, GCNs can learn powerful representations that capture the underlying properties of the graph.

Use Case: Molecular Property Prediction

In molecular property prediction, GCNs can predict properties like heat capacity, toxicity, or reactivity of molecules by learning from the molecular graph structure, where atoms are nodes and chemical bonds are edges. This approach leverages the inherent connectivity of the molecular structure to make accurate predictions.

Effectiveness of the Chosen Architecture:

The architecture chosen for this problem is a **Graph Convolutional Network** (GCN) with **two graph convolutional layers followed by a global sum pooling layer and two dense layers**. This architecture is effective for the following reasons:

Graph Representation: The GCN is well-suited for handling molecular data represented as graphs. The adjacency matrix and node features (atomic numbers) allow the model to capture the molecular structure's connectivity and properties.

Convolutional Layers: The use of graph convolutional layers (GCNConv) helps in learning local patterns and relationships between atoms within the molecule. This is crucial for understanding the molecular structure and predicting properties like heat capacity.

Global Pooling: The GlobalSumPool layer aggregates node features to form a graph-level representation, which is necessary for regression tasks where the target is a property of the entire molecule.

Dense Layers: The dense layers add non-linearity and enable the model to learn complex relationships from the pooled graph features, improving the prediction accuracy.

Model Performance: The architecture includes measures to evaluate performance, such as monitoring training and validation loss and mean absolute error (MAE). This helps in understanding the model's generalisation ability and tuning hyperparameters accordingly.

Libraries and Functions Used

The implementation utilises several Python libraries, including:

- **TensorFlow** for building and training the neural network.
- **RDKit** for handling molecular structures and operations**.**
- **NumPy** for numerical operations.
- **Spektral** components for creating datasets, defining graph convolutional layers, pooling layers, and data loaders.
- **Matplotlib** for creating plots.
- **Sklearn** function to split the dataset into training and validation sets.

Key functions and methods include:

- **Graph Convolutional Layers (GCL)**: Custom layers that perform convolution operations on graph data.
- **Model Definition and Compilation**: The GNN model is defined using TensorFlow's Keras API, compiled with an appropriate optimizer, and loss function for the task.
- **Training and Evaluation**: The model is trained on the dataset, and evaluation metrics such as accuracy are computed.

Execution of Code

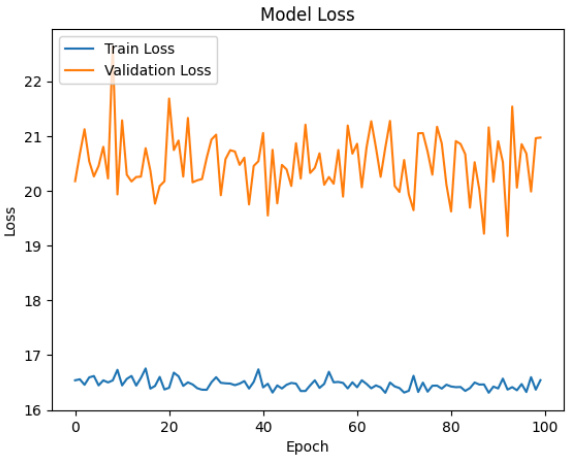
The Jupyter notebook is designed using Google Colab for better design of model and better availability of hardware resources on Colab platform. The code execution steps are mentioned below.

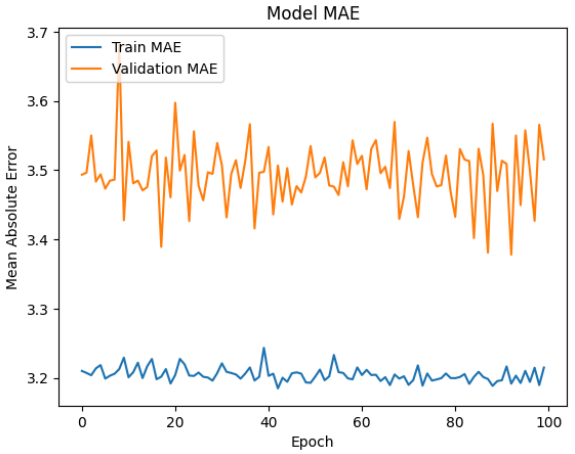
- **Environment Setup:** Install necessary libraries: tensorflow, spektral, pandas, rdkit, numpy, matplotlib
- **Upload the Dataset:** Upload the dataset from your local machine to Colab
- **Import Libraries:** Import all the required libraries
- **Load the Dataset:** Load the dataset (e.g., qm9_subset.csv) into a Pandas DataFrame
- **Explore the Dataset:**
  - Display the first 5 rows
  - Display the shape (number of rows and columns)
  - Display summary statistics
  - Display data types of columns
  - Check for null values
- **Convert SMILES to Graph:** Define a function to convert SMILES strings to graph representations
- **Prepare the Graph Dataset:** Create a custom Dataset class
- **Create Data Loaders:** Create data loaders for training and testing
- **Define the GNN Model:** Define the Graph Neural Network model with layer definition and activation function
- **Compile the Model:** Compile the model with a loss function and optimizer.
- **Train the Model:** Train the model using the training data, and validate using the validation set.
- **Plot Training History:** Plot training and validation loss and MAE
- **Predict Heat Capacity:** Functions to prepare a molecule and predict heat capacity
- **User Input for Prediction:** User input for a SMILES string and predict its heat capacity
- **Save Model:** Save the model weighted form.

Model Accuracy Description

The model was evaluated on both training and validation datasets, yielding the following performance metrics:

- **Training Loss:** 16.5444
- **Training Mean Absolute Error (MAE):** 3.2148
- **Validation Loss:** 20.9730
- **Validation Mean Absolute Error (MAE):** 3.5155





These results indicate that the model achieves a reasonably low MAE on both the training and validation sets, suggesting it can predict the heat capacity of molecules with an average error of about 3.2 cal/(mol K) on the training data and 3.5 cal/(mol K) on the validation data.

The relatively small difference between training and validation errors suggests that the model generalises well to unseen data. However, there is still room for improvement in terms of reducing both the training and validation errors.

Conclusion

In this project, we successfully implemented a Graph Neural Network (GNN) to predict the heat capacity of molecules using their SMILES representations. The workflow involved several key steps, from data preparation to model training and evaluation. Here's a summary of the process and the architecture used:

1. **Data Preparation:**
    - We started by loading a subset of the QM9 dataset, which contains molecular information and their associated heat capacities.
    - The dataset was explored for basic statistics, data types, and missing values to ensure data quality.
    - A function was defined to convert SMILES strings into graph representations, where each molecule is represented by an adjacency matrix and node features (atomic numbers).
2. **Custom Dataset Class:**
    - A custom dataset class, **QM9 Dataset**, was created to handle the conversion of SMILES strings to graph objects compatible with the Spektral library.
    - The dataset was split into training and testing sets to evaluate the model's performance.
3. **Graph Neural Network Architecture:**
    - We defined a GNN model using TensorFlow and Spektral. The model architecture consisted of:
    - Two graph convolutional layers (**GCNConv**) with **ReLU** activation functions to capture the local structure of the molecular graphs.
    - A global sum pooling layer (**GlobalSumPool**) to aggregate node features into a single vector representing the entire molecule.
    - Two dense (fully connected) layers to further process the aggregated features and output the final prediction.
    - The model was compiled with the Adam optimizer and a mean squared error loss function, with mean absolute error as an additional metric.
4. **Model Training and Evaluation:**
    - The model was trained using the training dataset with a defined number of epochs and batch size.
    - Training and validation loss, as well as mean absolute error, were plotted to monitor the model's performance over time.
    - Functions were provided to prepare individual molecules for prediction and to predict their heat capacity using the trained model.
5. **Prediction and Model Saving:**
    - An interactive input allowed users to provide SMILES strings, for which the model predicted the heat capacity.
    - Finally, the model weights were saved for future use without retraining the model.

Architecture Advantages

- **Graph Convolutional Layers:** These layers efficiently capture the relational structure of molecular graphs, allowing the model to learn from the connectivity and features of atoms in a molecule.
- **Global Sum Pooling:** This layer aggregates information from all nodes in the graph, providing a comprehensive representation of the molecule for the final prediction.
- **Spektral Library:** Leveraging the Spektral library facilitated handling graph data and implementing GNN layers, making the development process more straightforward.

Future Directions

- **Hyperparameter Tuning:** Experiment with different hyperparameters, such as learning rate, number of layers, and units per layer, to further improve the model's performance.
- **Extended Dataset:** Use a larger and more diverse dataset to enhance the model's generalisation capabilities.
- **Advanced Architectures:** Explore more advanced GNN architectures like Graph Attention Networks (GAT) or Message Passing Neural Networks (MPNN) for potentially better performance.

References:

- Chatgpt
- Deepchem
- Tensorflow docs
- Scikit-learn docs
- Kaggle dataset