Applying GAT for Drug-Target Interaction Prediction

Albert Guo, Adarsh Padalia, Yunzhe Qiao, Mark Wang

Wednesday, March 5, 2025

Problem Description and Research Question

Drug-Target Interaction (DTI) prediction is an essential process in the development of new drugs. However, the existing drug development process relies on experimental verification, which requires high costs and long development times. Recently, artificial intelligence (AI)-based molecular graph analysis has been playing an important role in the new drug screening process, and in particular, Graph Neural Networks (GNNs) have been attracting attention as a powerful tool for effectively learning the structural information of molecules.

The CandidateDrug4Cancer dataset contains 29 cancer-related target proteins, 54,869 drug molecules, and a total of 73,770 drug-protein interaction data. Molecules are expressed in the form of graphs, with atoms as nodes and chemical bonds as edges. The goal of this study is to analyze these molecular graphs and predict whether a specific drug is likely to have a significant biological interaction with a cancer target protein.

In this study, our objective is to verify whether Graph Attention Networks (GATs) show improved DTI prediction performance compared to existing traditional techniques (e.g., Morgan Fingerprints + XGBoost). We will implement a GAT with PyTorch. Furthermore, we will compare our results against existing molecular fingerprint-based techniques, and analyze whether the GAT model provides higher prediction accuracy. Lastly, we will graphically present the efficacy of a given drug (as evaluated by our GAT model).

Computational Plan

To implement this, we divide the computing process into the following steps. Data Processing, Graph Construction, Neural Network Model Architecture, Training Process, Evaluation, and Hyperparameter Tuning and Optimization.

- **Data Processing**

1. Dataset Structure

Typically, each row in the interaction table provides the following columns:

Column	Type	Description
`ChEMBL_ID`	String	ChEMBL identifier of the compound (e.g., `CHEMBL123456`).
`Target_ID`	String	ChEMBL identifier of the target (e.g., `CHEMBL1824`).
`pChEMBL_Value`	Float	$-\log_{10}$ of the measured IC_{50}.
`SMILES`	String	Chemical structure in SMILES format.
`Protein`	String	Full amino acid sequence of the target protein.
`Label`	Boolean	Binary indicator of active/inactive.

2. Equations for Label Computation and Molecular Features

2.1 pChEMBL and Activity Label

The dataset leverages a pChEMBL value, defined by:

$$p\text{ChEMBL} \;=\; -\log_{10}\big(\text{IC}_{50}\text{ in molar}\big),$$

which transforms IC_{50} from molar units into a logarithmic scale.

A threshold `pChEMBL` ≥ 7.0 (corresponding to $IC_{50} \leq 100$ nM) denotes a *potent* interaction. Hence, we define the activity label as:

$$\text{Activity\_Label} = \begin{cases} 1, & \text{if } p\text{ChEMBL} \geq 7.0, \\ 0, & \text{otherwise.} \end{cases}$$

2.2 Molecular Graph Features

For each molecule, we parse the SMILES string into a graph. Let $V = \{v_1, v_2, \ldots, v_n\}$ be the set of atoms, and $E = \{(v_i, v_j) \mid \text{bond exists between } v_i \text{ and } v_j\}$ the set of edges.

Each atom node v_i has features such as:

1. *Atom Type* (atomic number)
2. *Formal Charge*
3. *Degree* (number of covalent bonds to v_i)
4. *Hybridization* (e.g., sp, sp2, sp3)
5. *Aromaticity*

Each bond (v_i, v_j) has features like:

1. *Bond Type* (Single, Double, Triple, Aromatic)
2. *Conjugation*
3. *Ring Membership*

3. Data Preprocessing and Splitting

Step 1: Loading Data. We first read the CSV file that contain the compound–target interactions:

```
import pandas as pd

df = pd.read_csv("CandidateDrug4Cancer_full.csv")
print(df.head())
```

Step 2: Creating Activity Labels. If the label is not already present, we define:

```
df["Activity_Label"] = (df["pChEMBL_Value"] >= 7.0).astype(int)
```

Step 3: Splitting into Train/Validation/Test. We split randomly or by target:

```
from sklearn.model_selection import train_test_split

train_df, temp_df = train_test_split(df, test_size=0.3, stratify=df["Activity_Label"])
val_df, test_df = train_test_split(temp_df, test_size=0.5, stratify=temp_df["
    Activity_Label"])
```

4. Graph Construction (Feature Computation)

Using RDKit, we convert each SMILES to a molecule and extract the heavy-atom graph:

```
from rdkit import Chem
from typing import Any

def construct_molecular_graph(smiles_str: str) -> dict[str, list[Any]]:
    mol = Chem.RemoveHs(Chem.MolFromSmiles(smiles_str)) # remove explicit H atoms

    node_features = []
    adjacency_list = []
    edge_features = []

    for atom in mol.GetAtoms():
```

```
        feats = {
            "atomic_num": atom.GetAtomicNum(),
            "formal_charge": atom.GetFormalCharge(),
            "degree": atom.GetDegree(),
            "hybridization": str(atom.GetHybridization()),
            "aromatic": int(atom.GetIsAromatic())
        }
        node_features.append(feats)

    for bond in mol.GetBonds():
        i = bond.GetBeginAtomIdx()
        j = bond.GetEndAtomIdx()
        bond_feat = {
            "bond_type": str(bond.GetBondType()),
            "conjugated": int(bond.GetIsConjugated()),
            "ring": int(bond.IsInRing())
        }
        edge_features.append(((i,j), bond_feat))
        # Undirected adjacency
        adjacency_list.append((i, j))
        adjacency_list.append((j, i))

    return {
        "node_features": node_features,
        "edge_features": edge_features,
        "adjacency_list": adjacency_list
    }
```

Explanation: We record atom-level descriptors (atomic number, formal charge, etc.) and bond-level descriptors (bond type, conjugation, ring membership). The adjacency list is built from each bond pair (i, j).

- **Construct Graph Attention Network (GAT)**: The Graph Attention Network architecture first processes each drug's molecular graph through multiple GAT layers. Within each GAT layer, a node updates its embedding by computing attention scores for its neighbors. These scores highlight the relative importance of different neighboring nodes, and multi-head attention further stabilizes learning by combining insights from multiple parallel attention mechanisms. After the message passing phase, a pooling operation generates a single embedding for the entire molecule, typically by taking a mean or weighted sum of node embeddings. The final output of the GAT model is the predicted pChEMBL for the drug-protein pair.

  Remark: Graph Attention Network Equations

  - Attention Coefficients: For each node i, we compute attention scores a_{ij} toward each neighbor j as follows:

$$\alpha_{ij} \;=\; \frac{\exp\Big(\mathrm{LeakyReLU}\big(\mathbf{a}^{\mathsf{T}}\big[\mathbf{W}\mathbf{h}_i \,\|\, \mathbf{W}\mathbf{h}_j\big]\big)\Big)}{\sum_{k\in\mathcal{N}(i)} \exp\Big(\mathrm{LeakyReLU}\big(\mathbf{a}^{\mathsf{T}}\big[\mathbf{W}\mathbf{h}_i \,\|\, \mathbf{W}\mathbf{h}_k\big]\big)\Big)},$$

    where h_i is the current feature vector for node i, W is a trainable weight matrix, a is a learnable vector for the attention mechanism.

  - After normalizing coefficients, we update the node representation:

$$\mathbf{h}_i' \;=\; \sigma\Big(\sum_{j\in\mathcal{N}(i)} \alpha_{ij}\,\mathbf{W}\,\mathbf{h}_j\Big),$$

    where σ is a non-linear activation function, in this project, we will use ReLU, for convenience.

- **Training Process**: During training, the model uses Huber loss to predict whether a drug-protein pair is likely to bind. The Adam optimizer updates the network parameters, and regularization methods, such as dropout and weight decay, help the network generalize better. Early stopping is employed by monitoring the validation loss and halting training when performance ceases to improve, which avoids excessive overfitting.

- **Evaluation**

  1. Our study will compare the GAT-based approach against a baseline model that uses Morgan fingerprints to encode drug molecules. In the baseline method, circular fingerprints capture substructures around each atom, and a gradient boosting algorithm, like XGBoost, processes these representations together with simpler protein features. By evaluating the two approaches side by side, researchers can assess whether the structural awareness introduced by graph neural networks provides a significant performance boost over traditional fingerprint-based techniques.
  2. Model performance is evaluated using multiple metrics to capture different aspects of prediction quality. First, the predicted pChEMBL scores will be converted to `Labels`. Then, we will compute the precision and recall of the model. The correct prediction rate is the simplest metric, but AUC-ROC and AUC-PR often yield deeper insight, especially in cases with class imbalance. F1 score summarizes the balance between precision and recall, thus highlighting how effectively the model identifies correct interactions while minimizing false positives.

     Remark: Object-oriented Graph class implementation

     An object-oriented Graph class is implemented to simplify the process of storing and retrieving molecular structures. This class keeps track of adjacency lists, node features, and bond attributes. It also offers methods for adding nodes, adding edges, and retrieving neighbor information. By encoding all relevant data in a well-structured manner, the Graph class makes it easier to batch and feed molecules into the GAT. (see Listing 1)

     In addition, class `GATModel` encapsulating the GAT layers, along with fully connected layers for the final prediction. It will provide a `forward(drug_graph, protein_embedding)` method returning an interaction probability. Another `Trainer` class responsible for coordinating the training loop, including batch processing, loss computation, backpropagation, and metric logging will be implemented along with `GATModel` class (see Listing 2).

- **Hypermaterater tuning**: Hyperparameter tuning explores a range of values for the learning rate, hidden dimension, attention heads, and dropout rates. A systematic approach, such as grid or Bayesian search, helps identify settings that maximize validation performance while preserving generalization. Additional experiments can examine how the number of GAT layers affects the model's ability to capture higher-order structural relationships within each molecule.

- **Visualizing the Results**: If we are successful, we will have trained a model that is capable of determining the strength of a drug's biological interaction with a target protein. Given such a model, it is possible to determine the specific nodes and edges of the drug molecule that improve or impair the drug's efficacy using Feature Activation Maps. Given a drug/protein pair, we will construct a graphical representation of the drug molecule (using a library like `networkx`) that utilizes colour to show the relative contribution of different atoms and bonds to the efficacy of the drug.

Listing 1: Python implementation of a Graph class for molecular data

```
class Graph:
    """
    A Graph class to represent a molecular graph with atom-level and bond-level features.

    Instance Attributes:
        - neightbors: A dictionary mapping node IDs to a list of neighboring node IDs.
        - node_features: A dictionary mapping node IDs to a dictionary of that node's features.
        - edge_features: A dictionary mapping (node_i, node_j) to a dictionary of bond-level features.
    """
```

Listing 2: Scratch implementation of `GATModel` and `Trainer` classes

```
class GATModel:
    """
    A model class containing the multi-head GAT architecture and downstream fully connected layers for
        final prediction.

    Instance Attributes:
        - num_heads: Number of attention heads.
        - hidden_dim: Dimensionality of the hidden layers.
        - gat_layers: A collection of GAT layers.
        - fc_layers: Fully connected layers to combine graph embeddings with protein features for final
            prediction.
    """


```

```
class Trainer:
    """
    A class used for coordinating the training loop, including data loading, backpropagation, and logging.

    Instance Attributes:
        - model: The GAT-based model to be trained.
        - optimizer: The optimizer used for parameter updates.
        - criterion: The loss function for computing training loss.
        - metrics: A collection of metric functions for evaluation.
        - device: The device (CPU or GPU) on which training is performed.
    """
```

References

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980.*

Liao, Y., Gao, Y., & Zhang, W. (2023). Feature activation map: Visual explanation of deep learning models for image classification. *arXiv preprint arXiv:2307.05017.*

Raschka, S. (2014). An overview of general performance metrics of binary classifier systems. *arXiv preprint arXiv:1410.5330.*

RDKit. (2023). Open-source cheminformatics. *Retrieved from https://www.rdkit.org.*

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., & Bengio, Y. (2017). Graph attention networks. *arXiv preprint arXiv:1710.10903.*

Ye, X., Li, Z., Ma, F., Yi, Z., Li, P., Wang, J., ... & Xie, G. (2022). CandidateDrug4Cancer: An open molecular graph learning benchmark on drug discovery for cancer. *arXiv preprint arXiv:2203.00836.*