Appendix: Supplementary Studies

Graph Representation Learning

Contents

Chapter 1

Advanced Fragment-Based Walk for Molecular Docking Prediction

1.1 Introduction

Molecular docking prediction is an important step in finding new drugs. It helps researchers understand how ligands bind to target proteins. Accurate predictions can speed up the process of discovering new drug candidates. Traditional methods often assume that molecules are rigid, which does not fully capture how molecules move and interact in real life. To solve this problem, fragment-based methods have become popular. These methods allow scientists to study flexible molecular shapes and interactions.

This chapter introduces a new fragment-based method for molecular docking prediction. The method uses graphs to represent molecular fragments, random walks, and deep learning models to predict how proteins and ligands interact. By combining graph clustering, random walks, and graph neural networks (GNNs), the method provides a strong framework for analyzing molecular data and creating useful representations. The chapter explains how to implement different steps, such as graph clustering, generating random walks, and training models, and shows how these steps improve molecular docking predictions.

1.2 Methodology

The fragment-based walk method helps to predict how molecules interact. It uses graphs, random walks, and machine learning techniques to study molecular data and predict protein-ligand binding. This section explains each step of the method and why it matters.

1.2.1 Graph Clustering and Visualization

We begin with processing molecular datasets and constructing fragment graphs. This step is important for understanding the structural composition of molecules and identifying key fragments that may play a role in molecular interactions. The process is broken down as follows:

1. **Load Dataset:** We begin with loading a dataset containing SMILES (Simplified Molecular Input Line Entry System) strings into a pandas DataFrame. SMILES strings are a compact representation of molecular structures, which makes them ideal for computational analysis.

2. **SMILES Processing:** We use RDkit which is a tool for working with chemical data to convert each SMILES string into a molecular structure. RDKit helps read SMILES strings and create molecular objects that can be used for further steps. After the molecular structures are made, the molecules are broken into smaller pieces using the BRICS method. BRICS is a set of rules that splits molecules into smaller, meaningful parts. These parts are then used to build a graph where the nodes are the fragments, and the edges show how they are connected.

3. **Graph Visualization:** NetworkX library is used to visualize the fragment graph to gain insights. NetworkX is a Python package for creating, manipulating, and studying complex networks. Visualization helps to understand the connectivity and relationships between fragments, which is essential for subsequent analysis.

4. **Spectral Clustering:** We use spectral clustering to identify clusters within the fragment graph. First, we calculate the normalized Laplacian matrix of the graph and find its eigenvectors. These eigenvectors describe the graph's structure and serve as inputs for clustering. To group the nodes into two clusters, the KMeans clustering method is used. The clusters are visualized with different colors, providing a clear representation of the graph's community structure.

1.2.2 Random Walks and Graph Processing

Random walks technique is novel and powerful for exploring the structure of graphs and generating embeddings that capture the relationships between nodes. This step uses Node2Vec algorithm to generate the random walks, which is particularly effective for capturing both local and global graph structures. The process is detailed below:

1. **Random Walks (Node2Vec):** The Node2Vec algorithm is used to generate random walks on the fragment graph. Node2Vec introduces two parameters, p and q, which control the walk's behavior. The parameter p influences the likelihood of revisiting a node, while q controls the balance between exploring the graph locally and globally. The

`node2vec_random_walk` function is defined to generate random walks based on these parameters. Multiple random walks are generated for each node using the `generate_random_walks` function, ensuring comprehensive exploration of the graph.

2. **SMILES Processing:** The SMILES strings are processed to construct fragment graphs, as described in the previous subsection. These graphs serve as the input for the random walk generation process.

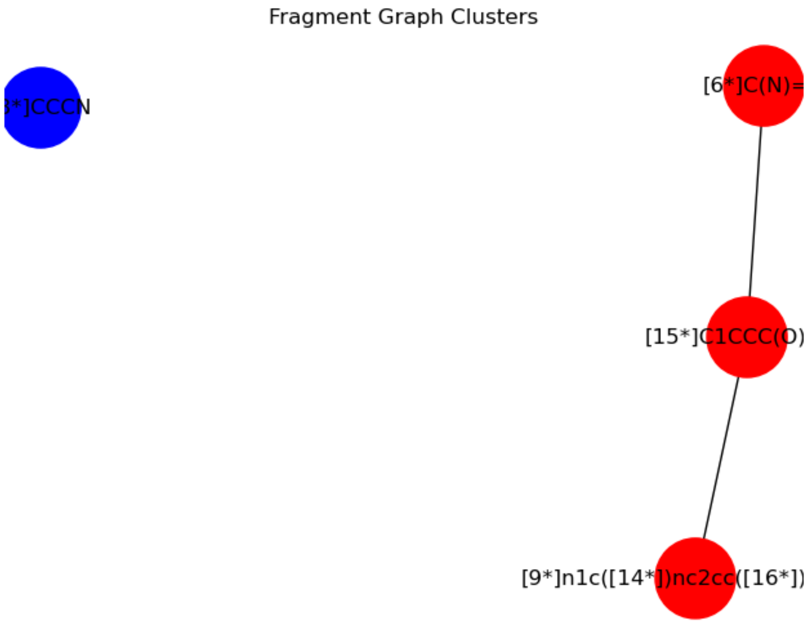

Figure 1.1: Example of a Fragment Graph Cluster

1.2.3 Model Building Process

The model building process combines the fragment graph construction, random walk generation, and deep learning techniques to predict protein-ligand interactions. This step is important for converting the structural data into predictions. The process is outlined as follows:

1. **Data Loading and SMILES Processing:** We load the dataset containing SMILES strings and convert the molecules into molecular objects using RDKit. Next, we fragment the molecules using BRICS, which generates a set of chemically meaningful fragments.

2. **Fragment Graph Construction:** We construct a graph where nodes represent the fragments and edges represent the bonds between them. This graph serves as the foundation for later analysis and model training.

3. **Random Walk Generation (Node2Vec):** We perform random walks with restart for each node in the fragment graph. These walks explore the graph's structure and generate sequences of nodes that capture the relationships between fragments. We generate multiple random walks for each node to ensure thorough exploration.

4. **Visualization:** We visualize the fragment graphs and the paths generated by the random walks using Matplotlib and NetworkX. We highlight specific fragments based on their importance or relevance to the dataset, creating a visual representation of the graph's key components.

5. **Word Embedding:** We treat the random walks as sentences and train a Word2Vec model to generate node embeddings. These embeddings capture the structural and relational information of the fragments in a low-dimensional vector space, making them suitable for machine learning tasks.

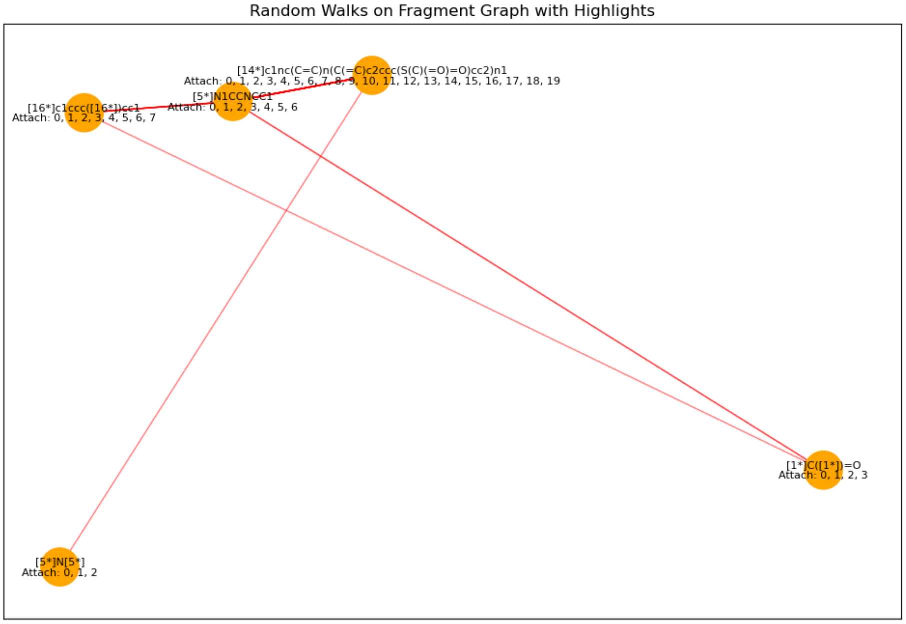
Random Walks on Fragment Graph with Highlights
[14*]c1nc(C=C)n(C(=C)c2ccc(S(C)(=O)=O)cc2)n1
Attach: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19
[5*]N1CCNCC1
Attach: 0, 1, 2, 3, 4, 5, 6
[16*]c1ccc([16*])cc1
Attach: 0, 1, 2, 3, 4, 5, 6, 7
[1*]C([1*])=O
Attach: 0, 1, 2, 3
[5*]N[5*]
Attach: 0, 1, 2

Figure 1.2: Example of the Random Walks on Fragment Graph

1.2.4 Fragment Graphs and Eigenvector Embeddings

This step creates eigenvector embeddings for fragment graphs, which help us mathematically represent the graph's structure. First, we load the dataset with SMILES strings and use RDKit to turn the molecules into molecular objects. Next, we break the molecules into smaller, meaningful pieces using BRICS. Then, we build a graph where the nodes represent the fragments and the edges represent the bonds between them, showing how the fragments are connected. After that, we calculate the normalized Laplacian matrix of the graph and find its eigenvectors, which give us a mathematical way to describe the graph's structure. Finally, we use t-SNE and PCA to reduce the data to fewer dimensions, making it easier to see patterns in the embeddings.

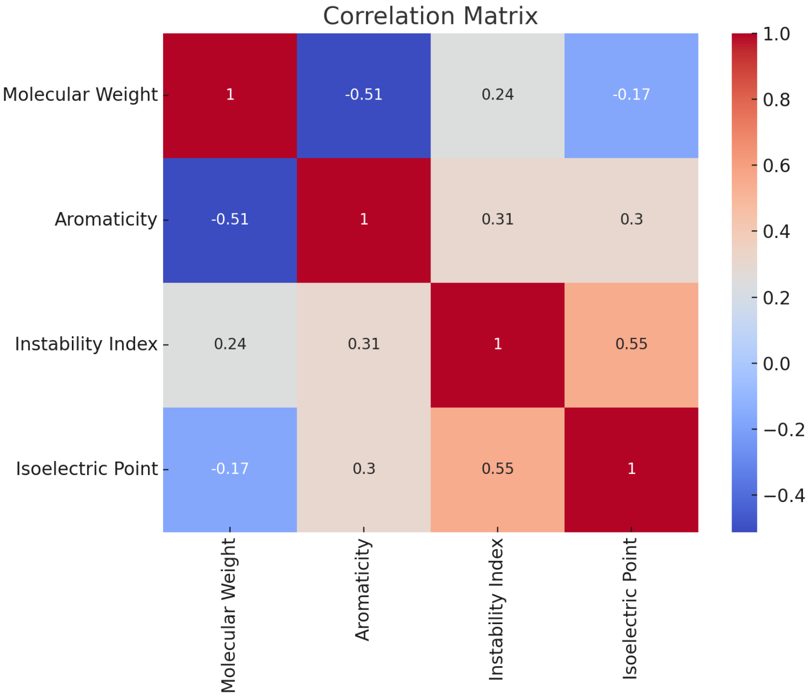

Figure 1.3: Correlation Matrix of different biochemical properties

1.2.5 Morgan Fingerprint Embeddings and Similarity Calculations

Morgan fingerprints are a popular way to capture the structure of molecules. In this step, we create Morgan fingerprints and calculate how similar they are. First, we load the dataset with SMILES strings and use RDKit to turn the molecules into molecular objects. Next, we break the molecules into smaller pieces using BRICS and create Morgan fingerprints for each piece, which describe the local environment of atoms in a molecule. Then, we calculate Cosine and Tanimoto similarities between the fragment embeddings to measure how close two fragments are based on their structure. We also add molecular properties like molecular weight to the embeddings to provide more context. Finally, we visualize the relationship between Cosine and Tanimoto similarities to better understand how the fragments are connected.

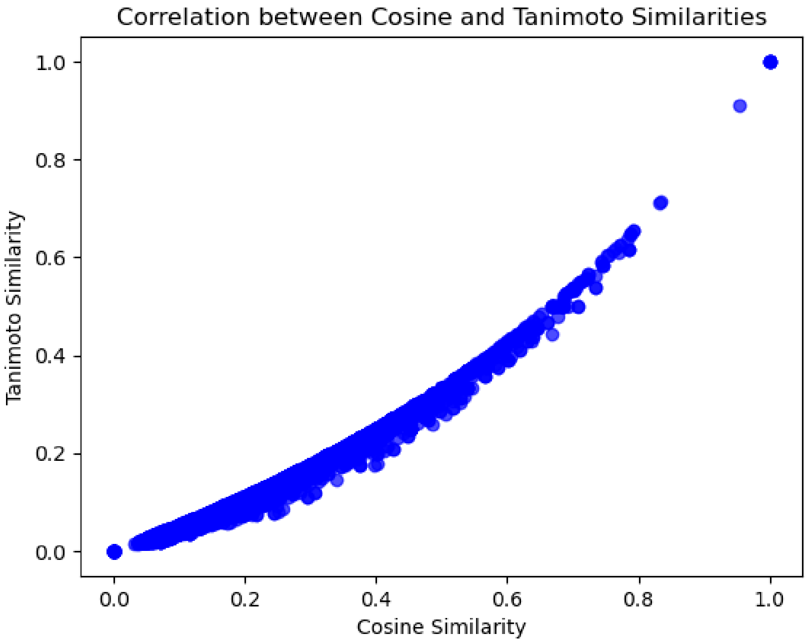

Figure 1.4: Tanimoto vs Cosine Similarity

1.2.6 Protein and Ligand Feature Extraction

Feature extraction is an important step for understanding the properties of proteins and ligands. In this step, we extract features from protein sequences and combine them with ligand features to get a complete picture. First, we load a dataset containing protein sequences and process them to extract useful features. Using the `ProteinAnalysis` class, we extract details like molecular weight, aromaticity, instability index, and isoelectric point. These features help us understand the physical and chemical properties of the proteins, which is essential for further analysis.

1.2.7 Combined Protein and Ligand Feature Extraction

In this step, we combine protein and ligand features into a single vector. This gives us a complete picture of how proteins and ligands interact. First, we load datasets containing protein sequences and SMILES strings. Next, we extract protein features, as explained earlier, and create random embeddings to represent the structure of the ligands. Finally, we combine the protein and ligand features into one vector for each protein-ligand pair. This combined vector helps us understand the interaction between the protein and ligand in detail.

1.2.8 Model Training and Evaluation

The final step in the methodology involves training and evaluating machine learning models to predict protein-ligand interactions. We train Graph Attention Network (GAT) models for this purpose. First, we define the models using `GATConv` layers for graph attention operations. We use global mean pooling and manual mean pooling to combine node embeddings. These models are designed to understand the relationships between nodes in the graph and predict interactions between proteins and ligands. Next, we prepare the data by loading the combined protein-ligand datasets and normalizing the features using `StandardScaler` to ensure they are on a consistent scale. Then, we train the models for 100 epochs using binary cross-entropy loss and the Adam optimizer, which helps the models improve their predictions. Finally, we evaluate the models by predicting the interaction probability between ligands and proteins using the sigmoid function. This gives us the probability of the protein ligand docking. The idea is to accept the docking if the probability of interaction is above a particular set threshold value.

1.3 Conclusion

The advanced fragment-based walk method described in this chapter offers a strong framework for predicting molecular docking. It uses graph-based representations, random walks, and machine learning models. However, the success of this method depends heavily on the quality and balance of the dataset used for training. In this study, a major problem was found: the dataset was completely imbalanced. It only included positive samples (examples of successful protein-ligand interactions) and no negative samples (examples of non-interacting pairs). This imbalance caused a big limitation in the model's performance.

When a dataset has no negative samples, the model learns to predict high probabilities for all inputs because there is no penalty for doing so. This happens because the loss function is minimized when the model always predicts the positive class, which is the only class in the dataset. As a result, the model cannot generalize and fails to tell the difference between interacting and non-interacting protein-ligand pairs. This issue was made worse by errors like the ZeroDivisionError, which occurred because of the lack of negative samples.

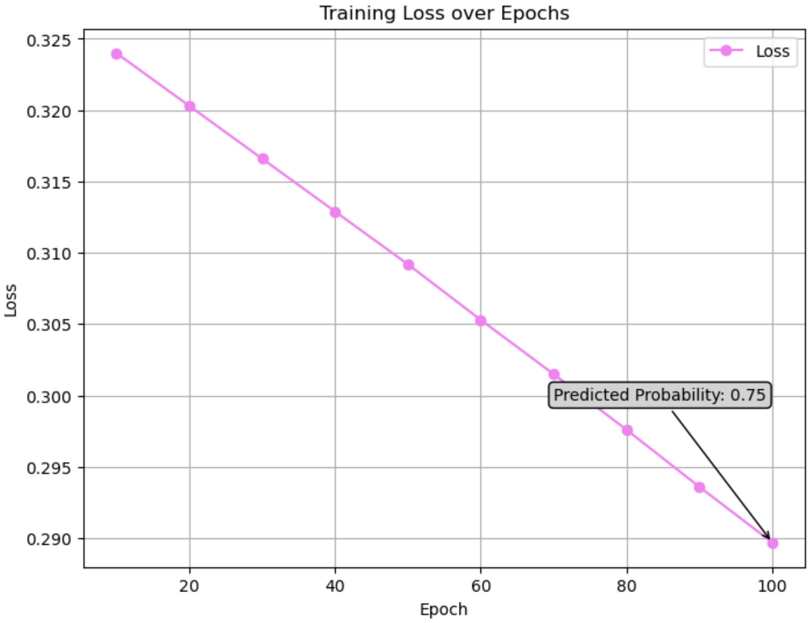

Figure 1.5: Model training with increasing Epochs

1.3.1 Steps to Address Dataset Imbalance

To improve the model's performance and address the dataset imbalance, the following steps can be taken:

1. **Introduce Negative Samples:** The most effective solution is to add negative samples to the dataset. If real negative samples are unavailable, synthetic data can be generated using techniques such as **SMOTE** (Synthetic Minority Oversampling Technique). Alternatively, domain knowledge can be used to manually create or label negative examples. Data augmentation, such as introducing transformations to existing data, can also help represent the negative class.

2. **Change the Loss Function:** Another approach is to modify the loss function to handle imbalanced datasets more effectively. For instance, using **Focal Loss** can help by reducing the penalty for easily classified examples and focusing on harder-to-classify cases. This can improve the model's ability to learn from imbalanced data.

1.3.2 Future Directions

Even though the current method gives good results, it is important to fix the problem of unbalanced datasets. This will help the model make better and more reliable predictions. In the future, we should focus on building datasets that have a good balance between positive and negative examples. We can also try methods to create more negative samples and test different loss functions that work well with unbalanced data.

To sum up, the method explained in this chapter is a good starting point for predicting molecular docking. But fixing the dataset imbalance is key. Once this issue is solved, the method can reach its full potential and play a bigger role in drug discovery and the study of molecular interactions.

Chapter 2

Solvent Accessibility-Guided Protein-Ligand Interaction Prediction

2.1 Introduction

Predicting how strongly a ligand binds to a protein (binding affinity) is a key challenge in drug discovery and structural biology. Traditional methods, such as molecular docking or analyzing full protein structures, are often slow and require a lot of computing power.

In this work, we try a simpler approach. Instead of using the entire protein, we focus on Solvent Accessible Surface (SAS) points—the areas on the protein surface that are exposed to the environment and where ligands usually bind. We represent these SAS points as nodes in a graph and connect them with edges based on how close they are to each other. Using this graph, we apply Message Passing Neural Networks (MPNNs), like Graph Attention Networks (GAT) and Graph Convolutional Networks (GCN), to learn local patterns that can help predict binding affinity.

The main question we explore is: Can we predict ligand-protein binding affinity using SAS-guided local environments, instead of relying on the full protein structure or docking methods? The following study aims to answer the question.

2.2 Methodology

Following is the methodology used to process protein-ligand interaction data, construct graph-based representations, and train machine learning models for link prediction. The process involves several steps, including data preprocessing, graph construction, feature extraction, and model training. Each step is explained in detail below.

2.2.1 Data Preprocessing and Graph Construction

We start by loading Protein Data Bank (PDB) files, which contain 3D structural information about proteins. These files are collected from a specific folder using the `glob` library, which finds all files with the `.pdb` extension. Each PDB file is then converted into a molecular object using RDKit, a tool for working with chemical data. The molecules are checked to ensure they have correct atomic structures and are fixed if any chemical errors are found. Molecules that fail this check are removed.

Next, we extract the 3D coordinates of the atoms from each valid molecule. These coordinates represent the surface-accessible points (SAS) of the protein, which are used to build the graph. We construct the graph by treating each SAS point as a node and connecting nodes with edges if they are within a certain distance (e.g., 6.0 Å). To efficiently find these connections, we use a KDTree, which quickly identifies pairs of points within the specified distance.

Each node in the graph is given features based on its 3D coordinates and local graph properties. For example, we calculate the degree of each node (the number of edges connected to it) and use it as a feature. Finally, we store the graph data, including node features, edge indices, and edge features (like distances between connected nodes), in a `Data` object from the PyTorch Geometric library. This object is used to train machine learning models.

2.2.2 Dataset Creation

We combine the processed graphs into a dataset using a custom `InMemoryDataset` class from PyTorch Geometric. First, we collect the graphs into a single dataset and save it to disk. The dataset includes all the necessary information, such as node features, edge indices, and labels. Next, we create a `DataLoader` to handle batching and shuffling of the dataset during training. This ensures that the model trains on random subsets of the data, which helps improve its ability to generalize.

2.2.3 Model Architecture

Two machine learning models are implemented for different tasks: a Variational Autoencoder (VAE) for graph representation learning and a Link Predictor for predicting protein-ligand interactions.

Variational Autoencoder (VAE)

The VAE is used to learn a compact representation of the graph data. It has three main parts: the encoder, the reparameterization trick, and the decoder. The encoder is a Graph Convolutional Network (GCN) with two layers. The first layer transforms the input features into a hidden dimension, and the second layer produces the mean and log variance of the latent space. The latent vector is then sampled using the reparameterization trick, which makes the sampling process differentiable during training. The decoder is a single GCN layer that tries to reconstruct the original input features from the latent vector. The VAE loss has two parts: the reconstruction loss, which checks how well the input features are reconstructed, and the KL divergence, which ensures the latent space follows a standard normal distribution.

Link Predictor

The Link Predictor is used to predict whether two nodes in the graph are likely to interact. It uses two GraphSAGE convolution layers to learn node embeddings by combining information from neighboring nodes. For each pair of nodes, the embeddings of the source and target nodes are combined to create edge features. A fully connected layer then predicts the probability of an interaction between the two nodes. The output is passed through a sigmoid function to produce a probability score. The model is trained using binary cross-entropy loss, which measures how close the predicted probabilities are to the true labels.

2.2.4 Training Process

We follow several steps to train the models. First, we load the dataset and split it into batches using the `DataLoader`. Each batch contains a small group of graphs, which we process in parallel. Next, we initialize the VAE and Link Predictor models with the correct dimensions for the input, hidden, and latent layers. We then train the models using the Adam optimizer, which adjusts the model parameters to minimize the loss function.

During training, we run a loop for a set number of epochs. In each epoch, we process every batch, compute the loss, and update the parameters using backpropagation. We also evaluate the model's performance using metrics like loss and accuracy. We log and print these metrics to monitor the training progress.

2.2.5 Handling Imbalanced Data

One of the challenges in training the Link Predictor is the lack of negative samples, which are examples of non-interacting protein-ligand pairs. We approach this problem by taking a few steps. We begin with generating negative samples by randomly pairing nodes that are not connected in the graph. These pairs are labeled as non-interacting. Next, we combine the positive and negative samples into one dataset to make sure the model is trained on a balanced mix of interacting and non-interacting pairs. Finally, we use binary cross-entropy loss to handle the imbalanced data. This loss function ensures that the model is penalized for incorrect predictions, whether they are positive or negative samples.

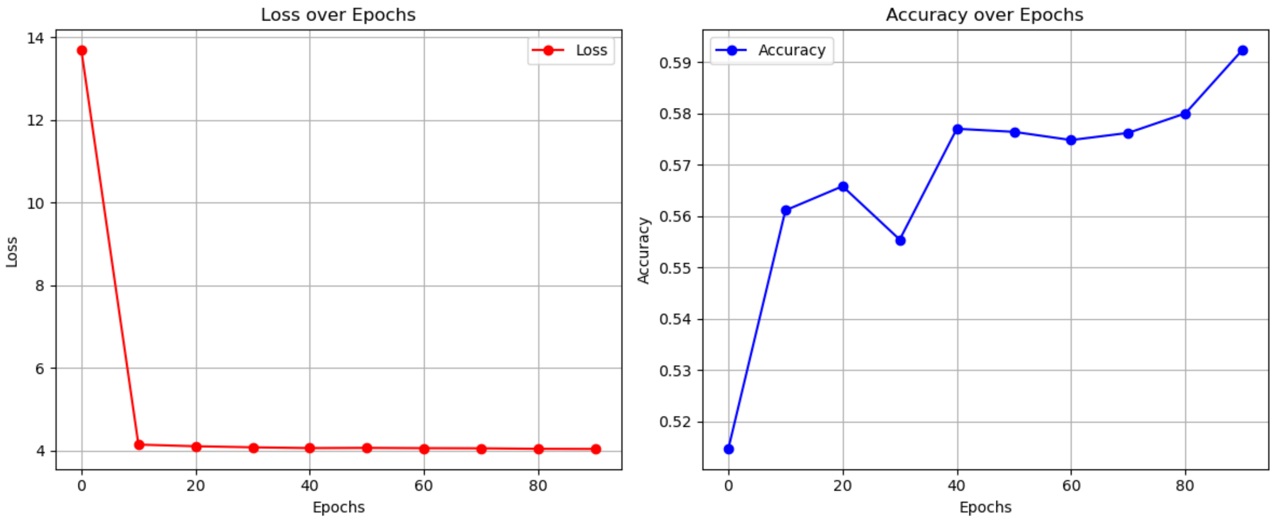

Figure 2.1: Loss and Accuracy over Epochs

2.3 Conclusion

In this work, we tried to develop a framework for processing protein-ligand interaction data, by creating graph-based representations, and training deep learning models. We used graph neural networks like GCNs and GraphSAGE to capture the structure and relationships in the data. We tried designing the training process to handle imbalanced data. This was our approach for predicting protein-ligand interactions, which can support drug discovery and molecular interaction studies.

However, the current model has limitations. It achieves only around 60% accuracy and lacks robustness. To address this, we need to train the model on larger datasets. The use of VAE to generate graphs may also introduce bias during training, affecting performance. In the future, we plan to improve the model by exploring techniques like data augmentation, using more diverse datasets, and testing advanced loss functions for imbalanced data. We will also consider incorporating domain-specific knowledge and using ensemble methods to combine multiple models for better predictions. These steps will help us build a more accurate and reliable tool for drug discovery and molecular interaction research.

Chapter 3

PINN for molecular docking

3.1 Introduction to Physics-Informed Neural Networks (PINNs)

Physics-Informed Neural Networks (PINNs) are a powerful class of machine learning models designed to solve problems involving Partial Differential Equations (PDEs). Unlike traditional methods, PINNs approximate PDE solutions by training a neural network to minimize a loss function that incorporates the PDE residual, initial conditions, and boundary conditions. This approach is mesh-free and does not require labeled data, making it highly versatile for both forward and inverse problems.

The core idea of PINNs is to integrate the governing physical equations into the neural network's loss function, effectively constraining the solution space to physically plausible outcomes. This is achieved by penalizing deviations from the PDE at collocation points within the domain. PINNs were formally introduced by Raissi et al. (2019) and have since gained significant attention due to their ability to handle nonlinear PDEs, such as the Schrödinger, Burgers, and Allen-Cahn equations. They are particularly effective for problems with complex geometries or high-dimensional domains, where traditional numerical methods struggle.

The concept of embedding physical laws into machine learning models is not entirely new. Early work by Dissanayake and Phan-Thien (1994) used neural networks to approximate PDE solutions, while Lagaris et al. (1998) extended this approach to irregular domains. Recent advancements in computational power, automatic differentiation, and open-source frameworks like TensorFlow have further accelerated the development of PINNs and their variants, such as Deep Galerkin Methods (DGM) and conservative PINNs (CPINNs).

PINNs offer several advantages over conventional methods. They are mesh-free, enabling on-demand solution computation after training, and provide differentiable solutions through analytical gradients. Additionally, they can simultaneously address forward and inverse problems, such as estimating model parameters from observational data, using the same optimization framework. This flexibility makes PINNs a valuable tool for scientific machine learning, with applications ranging from fluid dynamics to stochastic differential equations.

In this chapter we compare the performance of the Physics Informed Neural Network (PINN) with Graph Neural Network (GNN) in predicting the protein-ligand binding affinity.

3.2 Methodology

In this section we talk about the methodology used to predict protein-ligand binding affinity using Graph Neural Networks (GNNs). We begin the process with loading and preprocessing a dataset of protein-ligand complexes, computing molecular descriptors, and lastly training a GNN model, and evaluating its performance. Each step is explained in detail below.

3.2.1 Data Loading and Preprocessing

The first step involves loading and preparing the dataset, which includes protein-ligand complexes and their binding affinities. We start by loading the binding affinity data from the PDBBind database. This database contains protein-ligand complexes and their binding affinity values. We extract the binding affinity data from an index file, where each line includes a protein ID (PDB ID) and its corresponding binding affinity value. We skip entries with invalid or non-numeric affinity values.

Next, we load the ligand's SMILES representation from its SDF file for each protein-ligand complex. SMILES is a compact way to represent the molecular structure of the ligand. We then compute molecular descriptors for each ligand using RDKit, a tool for working with chemical data. These descriptors include molecular weight (`MolWt`), LogP (partition coefficient, `LogP`), topological polar surface area (`TPSA`), the number of hydrogen bond donors (`NumHBD`), the number of hydrogen bond acceptors (`NumHBA`), and a 2048-bit Morgan fingerprint, which captures the ligand's structural features. We combine these descriptors into a single feature vector for each ligand.

We remove ligands with invalid SMILES strings or missing descriptors to ensure the dataset is clean and reliable. Finally, we split the dataset into training and testing sets using an 80-20 split. We convert the feature vectors and binding affinity values into PyTorch tensors so they can be used with the GNN model.

3.2.2 Graph Neural Network (GNN) Model

We designed a basic GNN model to predict protein-ligand binding affinity using the computed molecular descriptors. The model takes the combined feature vector (molecular descriptors and Morgan fingerprint) as input. The size of the input layer matches the number of features in the vector.

The model has two hidden layers with 128 and 64 neurons, respectively. We use the ReLU activation function in these layers to add non-linearity to the model. The output layer has a single neuron, which predicts the binding affinity value.

To train the model, we use the Mean Squared Error (MSE) loss function, which calculates the difference between the predicted and actual binding affinity values. We optimize the model using the Adam optimizer to update the model's parameters during training. The idea was to keep the model as simple as possible.

3.2.3 Model Training

The GNN model is trained over 1000 epochs using the training dataset. In each epoch, the model makes predictions for the binding affinity values of the training data. After predictions, we calculate the Mean Squared Error (MSE) loss by comparing the predicted values to the actual binding affinity values. We determine how much each parameter in the model contributes to the loss using backpropogation. The Adam optimizer then adjusts the model's parameters to reduce the loss. To keep track of progress, we record the loss value for each epoch and print it every 100 epochs.

3.2.4 Model Evaluation and Visualization

Once training is complete, we test the model's performance on the test dataset. The model predicts binding affinity values for the test data, and we compare these predictions to the actual values. To understand how well the model learned during training, we plot the loss values over the 1000 epochs. The x-axis shows the number of epochs, and the y-axis shows the loss value. We also create a scatter plot to compare the predicted binding affinity values with the actual values. The x-axis represents the actual values, and the y-axis represents the predicted values. A diagonal line is added to the plot to indicate where perfect predictions would fall.

3.2.5 Summary

The method described in this section offers a basic framework for predicting protein-ligand binding affinity using GNNs. The process includes loading and preparing the dataset, calculating molecular descriptors, training a GNN model, and testing its performance. By using molecular descriptors and Morgan fingerprints, the model can capture the structural and chemical features of the ligands. The GNN architecture helps the model predict binding affinity accurately. The results from training and testing show that the approach needs to be significantly improved. Find the performance comparison of the GNN model with PINN model, towards the end of the chapter.

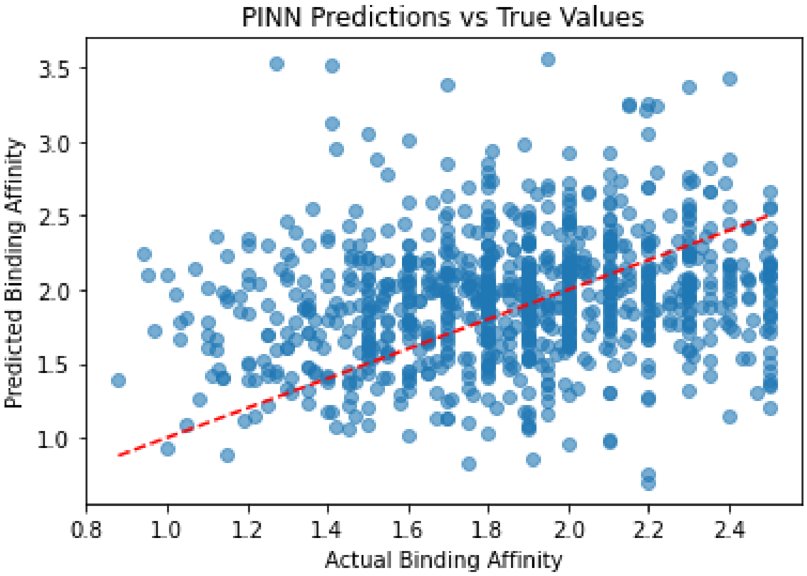

Figure 3.1: PINN which is a basic GNN

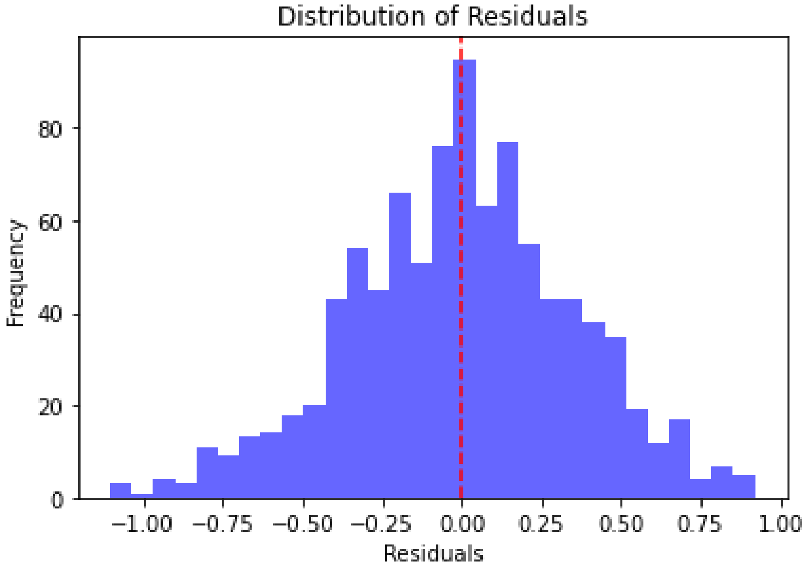

Figure 3.2: Distribution of Residuals for GNN

3.3 Methodology for Physics-Informed Neural Networks (PINNs)

Following is the methodology used to predict protein-ligand binding affinity using Physics-Informed Neural Networks (PINNs). While the data loading and preprocessing steps are similar to those used in the GNN approach, the PINN methodology incorporates physical laws into the model's loss function, enabling it to learn from both data and domain knowledge. Following are the steps:

3.3.1 Data Loading and Preprocessing

The data loading and preprocessing steps are identical to those described in the GNN methodology. The dataset is loaded from the PDBBind database, and molecular descriptors are computed for each ligand. These descriptors include molecular weight, LogP, topological polar surface area (TPSA), hydrogen bond donors (NumHBD), hydrogen bond acceptors (NumHBA), and a 2048-bit Morgan fingerprint. Additionally, three physics-based features are computed:

- **Molecular Mechanics Energy (MM_Energy):** A simplified approximation of the ligand's molecular mechanics energy, computed as $MolWt \times 0.1$.
- **Solvation Energy:** A proxy for solvation energy, computed as $-LogP$.
- **Entropy:** A simplified entropy term, computed as $TPSA \times 0.05$.

These physics-based features are concatenated with the molecular descriptors to form the input feature vector. The dataset is split into training and testing sets, and the features are standardized using `StandardScaler` to ensure consistent scaling.

3.3.2 Physics-Informed Neural Network (PINN) Model

1. **Model Architecture:** The PINN consists of a fully connected neural network with four layers:

   - An input layer with dimensionality equal to the number of features.
   - Two hidden layers with 256 and 128 neurons, respectively, using ReLU and LeakyReLU activation functions.
   - An output layer with a single neuron to predict binding affinity.

   Additionally, the model includes three learnable parameters (a, b, and c) that scale the contributions of the physics-based features (MM_Energy, Solvation_Energy, and Entropy) to the predicted binding affinity.

2. **Physics-Informed Loss Function:** The loss function combines the Mean Squared Error (MSE) between the predicted and actual binding

affinity values with a physics-based regularization term. The regularization term penalizes deviations from the physical relationship:

$a \cdot MM\_Energy + b \cdot Solvation\_Energy + c \cdot Entropy \approx PredictedAffinity.$

The total loss is computed as:

$Loss = MSE(PredictedAffinity, ActualAffinity) + 0.05 \cdot PhysicsRegularization.$

This ensures that the model's predictions are consistent with both the data and the underlying physical laws.

3. **Training Process:** The model is trained using the Adam optimizer. A learning rate scheduler reduces the learning rate by a factor of 0.5 every 200 epochs to improve convergence. The training data is divided into mini-batches of size 64, and the model is trained for 1000 epochs. The loss is tracked and printed every 100 epochs to monitor progress.

3.3.3 Model Evaluation and Visualization

After training, we test the model's performance on the test dataset. First, the trained model predicts the binding affinity values for the test data. Next, we plot the training loss history to see how well the model learned over time. The x-axis shows the number of epochs, and the y-axis shows the loss value. Finally, we create a scatter plot to compare the predicted binding affinity values with the actual values. A diagonal line is added to the plot to show where perfect predictions would fall. The x-axis represents the actual values, and the y-axis represents the predicted values.

3.3.4 Differences Between PINNs and GNNs

Following are the key differences between PINNs and GNNs:

- **Incorporation of Physical Laws:** PINNs explicitly incorporate physical laws into the loss function, making sure that the model's predictions are consistent with domain knowledge. GNNs, on the other hand, rely solely on data-driven learning and do not enforce physical constraints.

- **Loss Function:** PINNs use a composite loss function that includes both data-driven (MSE) and physics-based terms. GNNs typically use a purely data-driven loss function, such as MSE or cross-entropy.

- **Model Interpretability:** PINNs provide interpretable parameters (e.g., a, b, and c) that quantify the contributions of physical features to the predictions. GNNs, while powerful, are often treated as black-box models with limited interpretability.

- **Data Requirements:** PINNs can leverage small datasets more effectively by incorporating physical laws, whereas GNNs typically require larger datasets to achieve comparable performance.

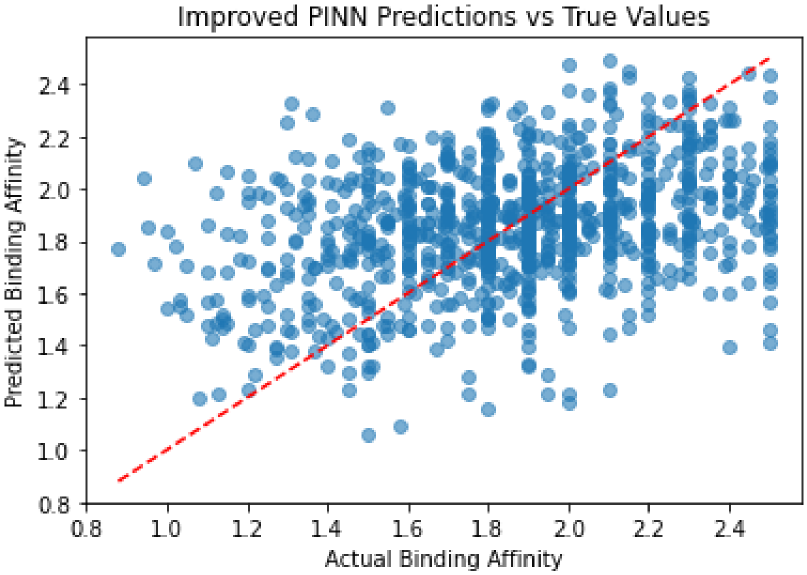

Figure 3.3: PINN vs True Values

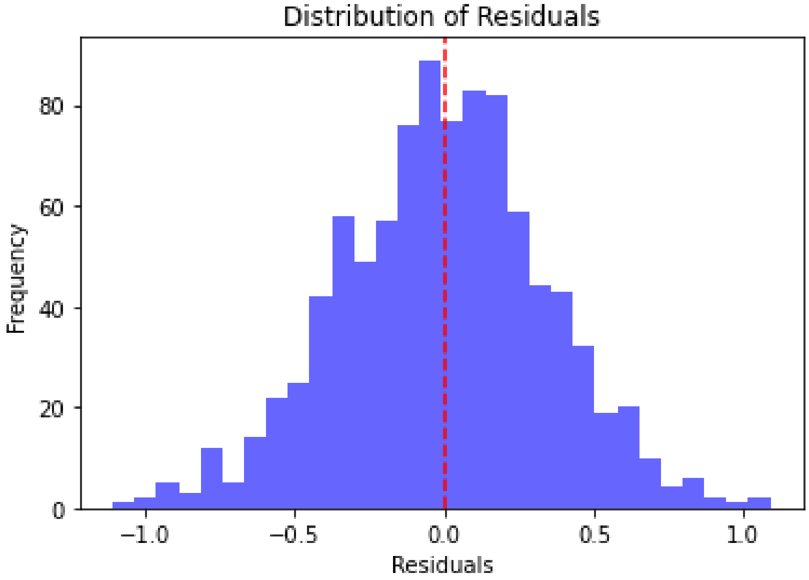

Figure 3.4: Distribution of Residuals for PINN

3.4 Conclusion

The method described in this section provides a complete framework for predicting protein-ligand binding affinity using Physics-Informed Neural Networks (PINNs). By including physical laws in the loss function, the PINN model makes predictions that are both data-driven and consistent with scientific principles. The use of physics-based features and regularization terms improves the model's interpretability and ability to generalize, making it a useful tool for drug discovery and studying molecular interactions. The training and evaluation results show that this approach works well, especially when compared to traditional GNNs.

For GNNs, the results were:

- MAE: 0.2782
- RMSE: 0.3528
- $\mathbf{R^2}$: -0.0367

For PINNs, the results were:

- MAE: 0.2726
- RMSE: 0.3447
- $\mathbf{R^2}$: 0.0106

The improvement in the $\mathbf{R^2}$ value for PINNs shows that incorporating physical laws helps the model make better predictions. This makes PINNs a stronger choice for tasks like drug discovery and molecular interaction studies.

It is important to note that this study used very basic neural network model. The goal was to compare a basic Graph Neural Network (GNN) model to a basic PINN model. The accuracy and results can be significantly improved by making the following adjustments:

Using advanced activation functions can help reduce validation loss. Choosing a more robust loss function, especially for datasets with outliers, can enhance performance. Techniques like learning rate scheduling or alternative optimizers can also lead to better results. Additionally, improving the model architecture by adding batch normalization, dropout, or increasing the depth of the network can further boost performance. These improvements can help make the models more accurate and reliable for predicting protein-ligand binding affinity.

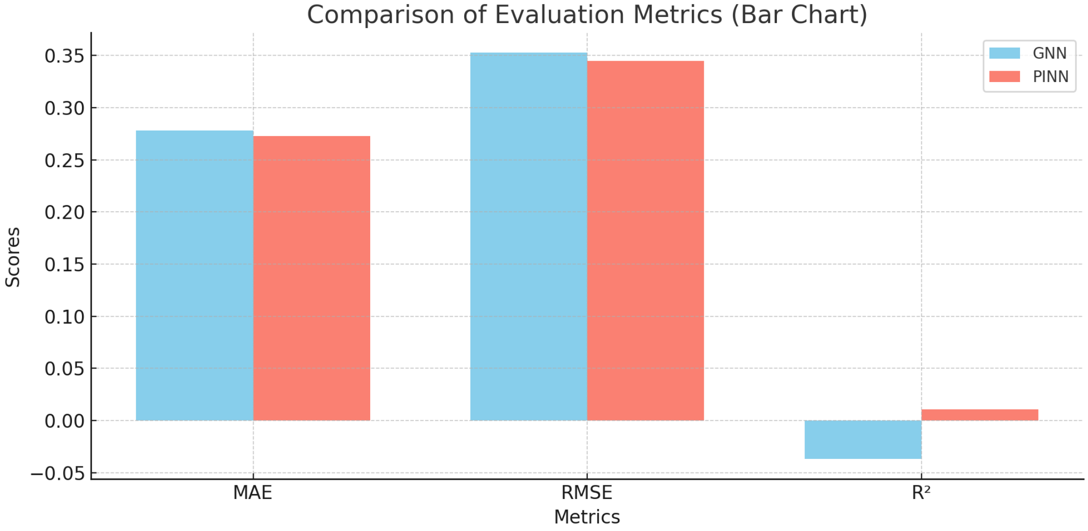

Figure 3.5: Evaluation Metrics (GNN vs PINN)