Molecular Graph Self Attention and Graph Convolution for Drug Discovery

by

Annamarie Elizabeth Bair

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2019

Author .....
Department of Electrical Engineering and Computer Science
August 23, 2019

Certified by .....
Peter Szolovits
Professor
Thesis Supervisor

Accepted by .....
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Molecular Graph Self Attention and Graph Convolution for Drug Discovery

by

Annamarie Elizabeth Bair

Submitted to the Department of Electrical Engineering and Computer Science
on August 23, 2019, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

Drug development is an important, but complicated and expensive process. By utilizing the power of deep learning, we aim to improve the current process of drug development. We model molecules as undirected graphs and use graph convolutions and self-attention to predict molecular properties. With a series of ablation studies, we demonstrate the added value of several key components in our network. We analyze two standard datasets: BBBP, which includes classification data on whether molecules pass the blood-brain barrier, and ClinTox, which includes toxicity information. Using our architecture, we are able to match state of the art performance on the BBBP prediction task.

Thesis Supervisor: Peter Szolovits
Title: Professor

Acknowledgments

First and foremost, many thanks to Matthew McDermott for his invaluable help throughout the course of this project. He built the initial core infrastructure and provided tremendous help in developing the main ideas in this thesis. This work would not have been possible without him. I would also like to thank Sam Finlayson and Elena Sergeeva for ideas and suggestions. Of course, thank you to Professor Pete Szolovits for giving me a place in your lab for the past two years and for providing guidance and support. Finally, thank you to my parents, my sister Zoe, and Eliza, Daysi, Sravya, Amir, Betty, and Chandler for the friendship, emotional support, and encouragement.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Drug discovery, the process by which candidate chemical compounds are identified as viable medications, is an inefficient and highly expensive process. Recent estimates indicate that it costs around $2.87 billion to produce a new drug and that only around 11% of candidate molecules end up being clinically approved [7].

In order to minimize this expense, computational methods for drug discovery have been developed and used in the pharmaceutical industry. One such method is virtual screening, which is used to determine which small molecules are most likely to bind to a specific target. In particular, quantitative structure-activity relationship (QSAR) models have been used to predict the biological activity or properties of a molecule based on its structure. This methodology has been used as a critical component of the pipeline to discover drugs to treat conditions including malaria, schistosomiasis (related to flatworms), tuberculosis, viral infections, and mood and anxiety disorders [15, 20].

In addition to existing computational chemistry methods, machine learning can offer benefits both in terms of time and cost required to identify new candidate drugs. Deep learning is showing incredible promise for applications in computational chemistry. In 2014, the winner of the Tox21 Data Challenge used deep learning for highly accurate toxicity prediction and demonstrated the viability of this type of method in chemistry applications [25]. Although deep learning is still in the process of being widely adopted by pharmaceutical companies, the historical success of other compu-

tational chemistry methods indicates promise for more advanced machine learning in the future. Here we use deep learning methods to predict various clinical properties of chemical compounds with the goal of improving the efficiency and reducing the cost of drug discovery.

1.1 Approach to problem

In this work, we use the chemical structure of molecules to predict various properties of compounds that are relevant to drug discovery.

Chemical structure can be encoded via a SMILES (Simplified Molecular-Input Line-Entry System) string, which contains ASCII characters that represent the connectivity of atoms and bonds in a given molecule. An example molecule and its corresponding SMILES string is shown in figure 1-1.

Figure 1-1: Molecule with its SMILES representation.

Using this molecular structure data, we use a deep learning model to create a biophysically meaningful embedding given a molecular structure. We then use this embedding to predict various attributes, including blood-brain barrier penetrance (BBBP) and clinical toxicity.

1.2 Contributions

We offer three main contributions in this work:

1. We introduce a novel deep learning architecture with graph convolutions, self-attention, and position embeddings that uses the graph structure of molecules

to perform prediction tasks.

2. We perform an ablation study to illustrate the added value of each network component.

3. We match state of the art classification performance on the BBBP prediction task.

Chapter 2

Background

The literature relevant to this work falls into two main categories: the work related to machine learning for chemistry and the work discussing network architectures, including graph convolution and self-attention. First, we will discuss the background in machine learning for chemistry tasks. Then, we will introduce graph convolutional networks and self-attention and discuss architecture augmentations used in similar work.

2.1 Machine Learning for Drug Discovery and Chemistry

Two main papers have previously approached this problem and established baseline results. One key paper was that of Wu et al., which introduces MoleculeNet. MoleculeNet is a large and comprehensive benchmarking dataset. They compiled datasets organized into four main categories: Quantum Mechanics, Physical Chemistry, Biophysics, and Physiology. In each of these categories are a handful of datasets, each of which includes one or more regression or classification tasks. This compilation is critical for advancing machine learning for chemistry, as standardized datasets allow for results to be more easily compared. This work also includes a set of preliminary results using various standard and graph-based models. They implement several

models in an open source package, DeepChem. These include several basic models such as logistic regression, support vector machines, and random forests, as well as graph based methods, such as graph convolutional models, a weave model (similar to a convolutional model except incorporates information from all nodes at each step), and a message passing neural network model. This package additionally has code for splitting the dataset in a consistent manner, again allowing for easier reproducibility. MoleculeNet is highly relevant to this work since it provides us with standard datasets to use [30].

Another key paper is Chemprop [31]. This work also builds upon MoleculeNet, focusing on the graph-based methods and on improving performance. They develop the directed message passing neural network (D-MPNN) architecture and establish new state of the art results on several prediction tasks. The code is released publicly as part of the Chemprop framework, which also allows for easy prediction on all MoleculeNet datasets. This was our main comparison point, as their command line interface made it easy to compare their D-MPNN model against ours on identical splits of the MoleculeNet datasets.

A variety of other previous work has demonstrated the feasibility of using deep learning methods to predict target tasks based on chemical structure data [30, 27, 24]. [27] uses chemical structure, drug target, and side effect information to predict drug-disease interactions. [24] predicts potential drug repositioning based on chemical structure, in addition to drug target and genetic similarity. Finally, [28] uses gene expression and chemical structure data to directly predict adverse drug reactions.

2.2 Graph Convolutional Networks

2.2.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) have achieved remarkable success in recent years on a variety of tasks, including speech, image, and video recognition [2, 12, 16, 17]. Their power comes from their ability to take advantage of spatial and temporal

dependencies in the input. In images, for instance, patterns of adjacent pixels provide information about shapes and patterns, which in turn provide information about the contents of the image. CNNs use a hierarchical series of convolution and pooling operations in order to extract higher-level features from the input data. The two significant components of CNNs are *convolutional layers* and *pooling layers*.

Convolutional layers apply a filter to patches of the input pixels and output a convolution of the input with the filter. At a given layer, the same filter is applied to patches spanning the entire image. This translation invariance is a key property of convolutional networks [17].

Pooling layers aggregate the output of a previous layer and perform a reduction in the size of the data. Examples of this type of aggregation include max pooling, where the input is partitioned and the maximum value of each partition is outputted, and average pooling, where the average value over each partition is outputted [17].

Figure 2-1 shows an input on the left organized into smaller receptive fields. A convolution or pooling operation can then be applied to each square, and the output is shown on the right.

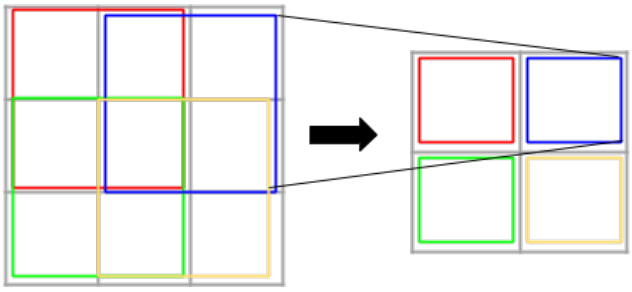

Figure 2-1: Schematic of pooling and convolution receptive fields.

2.2.2 Graph Convolutional Networks

Despite their power in a variety of tasks, CNNs are limited to inputs that have a homogeneous structure, such as images in which each interior pixel has eight neighbors

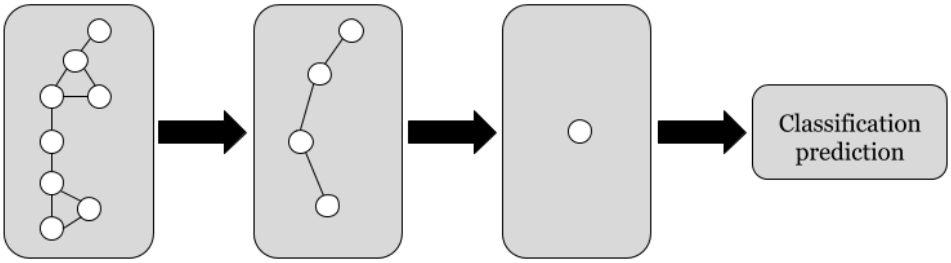

Figure 2-2: Schematic of a graph convolutional network. Through processing, the network reduces the input graph until it is able to make a classification prediction.

(edges can be accounted for in a similarly structured manner). Graph Convolutional Networks (GCN) extend the concepts of CNNs and add flexibility to allow for processing of arbitrary graphs.

There are a few additional challenges with working with graphs instead of more structured inputs. Defining a convolution operation is hard. Since nodes in a graph can have arbitrary degree, the same filter cannot be applied to each node. Also, the pooling operation is hard on graphs. Finding cliques in a graph is NP-Hard and so pooling cannot as easily be defined as for more homogeneously structured inputs [11]. Some approaches to these problems involve using spectral graph theory to define parameterized filters. These filters are conceptually similar to convolutions [4, 14]. [5] develops a method of fast localized spectral filtering which parallels the filtering operation done on images, but accounts for arbitrary graphs which lack the structure of pixels within an image. Other approaches are to design domain-specific architectures. [8] works specifically on applying GCNs to learn molecular fingerprints, which combine information from each node (atom) in the molecule to create a vector representation of the entire molecule. [13] uses graph convolution to create a more flexible representation of a molecule than the molecular fingerprint described in [8].

2.2.3 Representing a Molecule as a Graph

We use a graph to represent the atoms and bonds and relationships present between them. As a simple example of this representation, we can use the carbon dioxide, or

CO_2 molecule. Carbon dioxide consists of a central carbon atom, connected to two oxygen atoms, each with a double bond. In our graph representation, we represent each atom or bond as a node. Thus, our graph of carbon dioxide has five nodes: the first oxygen, the first double bond, carbon, the second double bond, and the second oxygen. The connectivity pattern can be observed both in the schematic of the molecule and in the graph representation in figure 2-3.

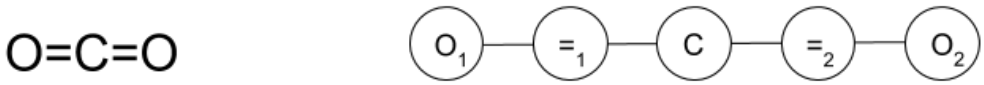

Figure 2-3: The left side shows the schematic representation of carbon dioxide; the right side shows a way of representing the molecule as a graph.

2.3 Self-Attention

Convolution is inherently a local operation: the output of a convolutional layer is a function of a node and its neighbors. In contrast, self-attention is a mechanism by which a network can capture long-range interactions by processing elements based on their similarity, rather than by their proximity to other elements. Attention uses a similarity function and then updates a given node as a weighted sum of other nodes based on similarity. We compare how similar the embeddings are for all pairs of nodes, and then update each embedding to be a weighted combination of other node embeddings, with a higher weight for more similar embeddings. Self-attention was popularized in [26]. In this paper, the authors used self-attention to achieve state of the art performance on a machine translation task. They introduce scaled dot-product attention, which computes similarity between vector representations of different elements. This work inspired other papers, which we follow more closely in the implementation of our network.

One such paper was [23], in which the authors use both self-attention and relative position representations to perform machine translation tasks. While attention

captures similarity between elements, this relative position representation captures sequential distance between elements. The addition of relative position representations improved performance in machine translation over attention alone.

[3] describes combining convolutions with self-attention. Their architecture computes attention and convolution in parallel and then concatenates the outputs of the two sets of layers. This work differs from ours in that they use CNNs to work with images rather than our GCNs to work with graphs, and that they compute attention and convolution in parallel, rather than in series as we do, but the general concepts translate well. They evaluate their model on ImageNet classification and find that the combination of attention and convolutions outperforms a purely convolutional network.

2.3.1 Inspiration from Natural Language Processing

As described above, self-attention was developed for a machine translation task. BERT is one such Natural Language Processing (NLP) model that achieved state of the art results using self-attention and has in turn inspired other models, including RoBERTa, XLNet, and ERNIE [6, 18, 33, 32]. The idea of applying NLP methods to computational chemistry is not new. NLP techniques have been used to analyze the SMILES representations of molecules [22, 10]. Although in this work we did not primarily work with the SMILES representation, we found that NLP methods proved useful in analysis of the graph representations of molecules.

Chapter 3

Data

3.1 Datasets

The primary source of data are the datasets compiled by MoleculeNet [30]. MoleculeNet consists of four main categories of datasets: Quantum Mechanics, Physical Chemistry, Biophysics, and Physiology. We restrict ourselves to the Biophysics and Physiology datasets, and in the current work, focus on two Physiology tasks: BBBP and ClinTox. While the tasks in the Quantum Mechanics and Physical Chemistry datasets are all regression tasks, most of the Biophysics and Physiology tasks are classification. The Biophysics datasets we use are: PCBA, MUV, HIV, and BACE; the Physiology datasets we use are BBBP, Tox21, ToxCast, SIDER, and ClinTox. Table 3.1 describes the information in each of these datasets, including the number of prediction tasks and the number of chemical compounds present in each dataset.

Due to computational constraints, as described in Chapter 5, we only report results here on BBBP and ClinTox. However, our methods and architecture easily generalize to all datasets described here.

3.2 Splits

[30] also suggested different ways of splitting the datasets into training, validation, and test sets. Some of the tasks use random splitting, but many use scaffold splitting.

Dataset	Description	# Tasks	# Compounds
BBBP	Whether a drug can penetrate the membrane which separates neurons from circulating blood and reach its intended target	1	2039
ClinTox	FDA Approval status and failure in clinical trials due to toxicity	2	1478
PCBA	PubChem BioAssay: understand biological activities of small molecules	128	437929
MUV	Maximum Unbiased Validation: validate virtual screening techniques	17	93087
HIV	Ability of compounds to inhibit HIV replication	1	41127
BACE	Binding affinity for human β-secretase 1 (BACE-1) inhibitors, related to neuronal myelin sheath formation	1	1513
Tox21	Toxicity measurements on 12 targets	12	7831
ToxCast	Toxicology data from *in vitro* high throughput screening	617	8575
SIDER	Side Effect Resource: describes adverse drug reactions	27	1427

Table 3.1: Prediction targets

Scaffold splitting first groups together the molecules which share similar 2D molecular structure, or scaffold. Then, all molecules that share a scaffold are placed into just one of training, validation, or test. This is a harder classification task since molecules in each of the subsets are more structurally different. However, this is also more realistic and robust, as it allows us to understand how models will perform on new molecules with new molecular scaffolds. In order to compare our results as faithfully as possible to previous work, we use the splits shared by [31] on their github repository. Specifically, for BBBP, we use a scaffold split and for ClinTox we use a random split. These splits divide up each dataset into 80% train, 10% validation, and 10% test data. For each task, there are thirteen different splits.

When tuning hyperparameters, we train each model on training data and then evaluate it on validation data. We choose the hyperparameters that lead to the highest validation performance and then use these to train a model (using combined train and validation sets) which is evaluated on the test set. These test set results

are reported as our final performance metrics. This strategy of splitting the data into train, validation, and test sets prevents us from over-fitting to the training data and gives the best chance of performing well on the single run of the test set.

3.3 Dataset Balance

Imbalanced datasets can pose a challenge for classification problems. All of the datasets from MoleculeNet are relatively imbalanced. Table 3.2 shows the balance of each of the prediction tasks that we focus on in the present work.

Dataset	**Prediction task**	**Percent Positive Samples**
BBBP	Pass blood-brain barrier	76.51%
ClinTox	FDA approved	93.64%
	Failed clinical trial	7.58%

Table 3.2: Dataset balance

Chapter 4

Methods

4.1 Pipeline

There are two main components to our pipeline: the drug embedder, which takes in a molecule and produces a unique vector embedding, and the classifier, which predicts properties of each drug based on its embedding. When training our model, these components work together to ensure that the vector embedding of the molecule contains information relevant to the prediction task. A schematic of the overall pipeline is shown in figure 4-1.

The entire pipeline was implemented in Python with PyTorch. Other common libraries such as Pandas, Scipy Stats, Numpy, and Scikit-Learn were used for various elements of data processing and prediction.

4.2 Preprocessing

The first component of our pipeline involves preprocessing the data. The datasets were downloaded as CSV files where each line contains a SMILES string and then 0s or 1s to indicate values of the various attributes recorded in the dataset. Before running the algorithms, we create several matrices that will be used for various versions of the graph convolutional network. These include adjacency matrices, 3D distance matrices, and relative position matrices.

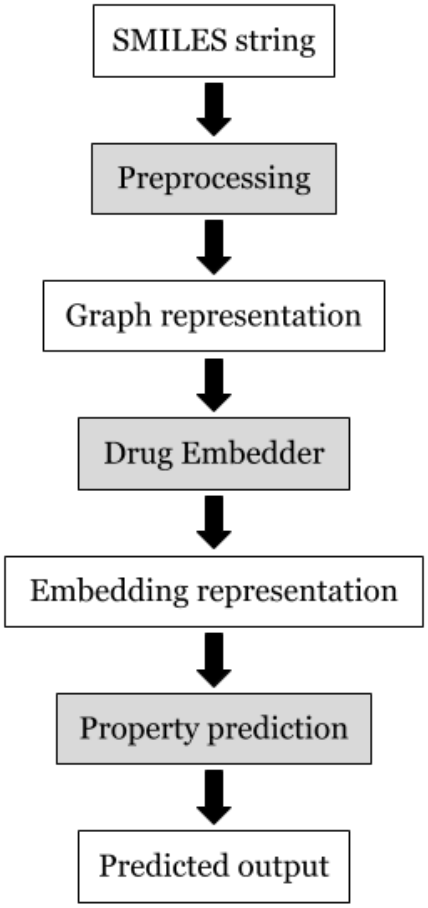

Figure 4-1: Schematic of the property prediction pipeline. Grey boxes are processing components and white boxes indicate the format of the data.

To begin, we parse the SMILES strings using RDKit. RDKit is an open source package commonly used for computational chemistry and analysis of molecules [1]. We first use RDKit to canonicalize, or avoid duplicate representations of the SMILES. Of most use in this project, the API can take in a SMILES string and return the atoms and bonds and their connectivity for the given molecule. From this information, we can create a graph representation of the molecule. We represent both atoms and bonds as nodes, and the connections between them as edges. Encoding bonds as nodes allows for our model to encode more detail about the molecule, such as whether the molecule contains single or double bonds connecting various atoms. This graph representation allows us to compute useful quantities in processing the molecules. We

use these connectivity patterns to create an adjacency matrix for each molecule.

4.3 Embedding

Before discussing the specifics of the embeddings used in this network, we will first take a short detour to discuss the nature of embeddings and why they are useful. An embedding matrix maps integer indices to randomly initialized vectors of a specified dimension, as shown in figure 4-2.

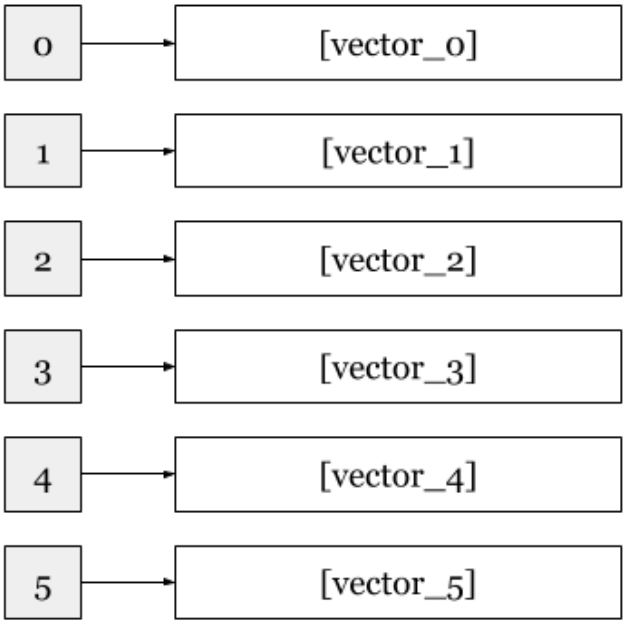

Figure 4-2: An embedding matrix maps index values to vectors. Since the mappings remain constant but the vectors change over time, embedding matrices are good ways to learn and store representations of items.

This embedding methodology of representing objects by vectors was first introduced in the field of Natural Language Processing (NLP) via Word2Vec [19]. Embeddings which evolve through unsupervised learning emerged in works such as GLoVE, also a NLP model [21]. More recently, many other types of embeddings, including molecular embeddings have also been developed [29].

As a whole, our network takes in individual embeddings that represent components of a molecule and processes these to produce a final embedding vector that represents the entire molecule. This final vector is used to perform a prediction task. The two initial embedding types we use are:

1. Vectors to represent each type of atom or bond.

2. Vectors to represent the distance (3D or relative position) between atoms and bonds in the molecule.

We begin with randomly initialized vectors for each atom type, bond type, and distance. We pass these embeddings into our network, which processes them as specified and produces a prediction for the molecule as a whole. During backpropagation, the embeddings will update via stochastic gradient descent to minimize the loss during the training run. Over time, these embeddings will come to encode useful information about the prediction task. Later sections in this chapter describe in greater detail how these vectors are processed as they are passed through the network.

4.4 Drug Embedder

Our drug embedder included graph convolution layers and self-attention layers. Combinations of these layers along with standard ReLU and dropout allowed us to take advantage of both global and local features of the molecule. We will discuss the details of each of these layers in turn.

4.4.1 Graph Convolutions

Similar to convolutional neural networks, we define convolutions and pooling over a graph.

The graph convolution layer takes as input a set of adjacency matrices which represents molecular graphs, and outputs embeddings of each graph. The implementation of the layer can be simplified due to the constrained nature of the molecular graphs. In order to represent each of these atoms and bonds numerically, we assign

an initial random embedding vector to each atom or bond type. This embedding allows for a type of invariance in analyzing the molecular structure: a carbon atom located anywhere in the molecule will have the same representation. Over time, these embeddings change so as to characterize important properties in the molecular graph which can be used for a particular predictive task.

Next, we apply a type of convolution over the molecular graph. We observed that due to the relatively constrained structure of molecules, the maximum degree of any of our graphs is 4. Figures 4-3 and 4-4 show the distribution of number of neighbors of each node in each dataset. This constraint allows us to depart from spectral and fingerprint methods used in [5] and [8].

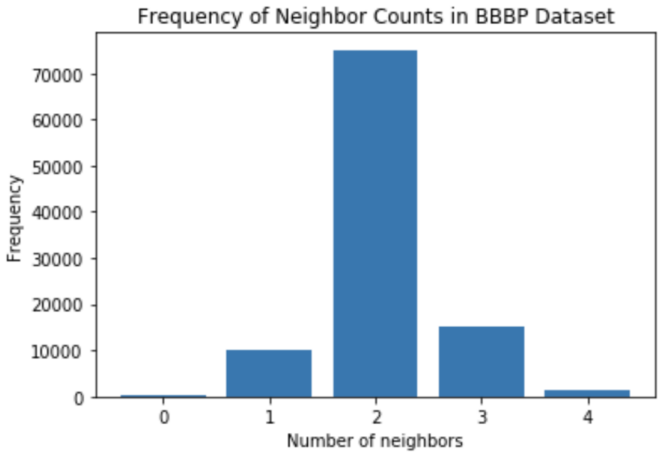

Figure 4-3: Degree distribution of atoms and bonds in BBBP dataset.

This maximum degree constraint allows us to create different functions that operate on each node based on the degree of that node. Specifically, we create four functions f_1, f_2, f_3, f_4 where f_i is applied to nodes with i neighbors. These functions concatenate the embeddings of each node and its neighbors and pass it through a learned linear layer. This performs the operation of convolving a node with its neighbors, with the invariant property that identical segments of molecules will be processed identically after being passed through the convolutional layer. However, this invariant property requires that the input ordering of the neighbors be preserved. In our work, these (up to four) neighbors are ordered in a consistent and arbitrary man-

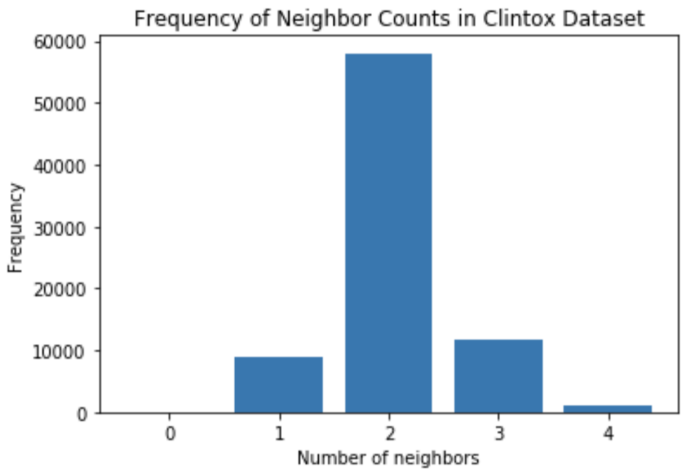

Figure 4-4: Degree distribution of atoms and bonds in ClinTox dataset.

ner as determined by RDKit. This ordering does not necessarily allow the invariant property to hold throughout the molecule. In addition, some arbitrary ordering information may be used by the network, which is not ideal. While there is no explicit representation for subgraphs within the molecule, this convolution information is still stored: as vectors representing the nodes pass through the network, they are updated to include the information about their neighbors. For example, if there are multiple locations in a molecule where a carbon atom is connected to two aromatic bonds, each carbon atom will have the same output after the first convolutional layer, since the nodes themselves and their neighbors are all identical. However, as the network progresses through additional layers, different sections of the molecule will begin to diverge as they begin to incorporate information from broader neighborhoods through repeated convolution.

Finally, we perform pooling over the entire graph. This collects the various processed embeddings into one vector that represents the entire graph.

4.4.2 Self Attention

Attention works on a graph by performing a series of operations on the embedding vectors that represent each node in order to find certain patterns of connectivity throughout the graph. Suppose we have a molecular graph $G = (V, E)$ as our input,

where V is the set of vertices and E is the set of edges. Each vertex in the graph is represented as an embedding vector x_i of dimension d_x. The goal of a self-attention layer is to have similar outputs for similar input nodes in the graph. Here, since we use a combination of convolution (which combines a node with its neighbors) and self-attention, we can generally think of "similar" as referring to the identity of a node and its connectivity pattern with its neighbors. In order to determine this similarity, we use the scaled dot-product method introduced by [26]. The scaled dot-product is an operation we can apply to each node in the graph to determine its similarity with each other node and then re-weight the node in terms of similar nodes accordingly. The equation to compute attention for node i is:

$$\text{attention\_output}_i = \sum_j \text{softmax}\left(\frac{(z_q)(z_k)^T}{\sqrt{d_z}}\right) x_j W_v \tag{4.1}$$

where

$$z_q = x_i W_q \tag{4.2}$$

and

$$z_k = x_j W_k \tag{4.3}$$

$W_q, W_k, \text{and } W_v \in \mathbb{R}^{d_x \times d_z}$ are all learned linear transformations that are applied to node embeddings.

4.4.3 Relative Position Embeddings

As in [23], we also added relative position embeddings to our attention layers. We used the adjacency matrix as a basis for creating a relative position matrix. In our implementation, we define the distance between two nodes in the graph to be the shortest path length between pairs of molecules in our graph, with a maximum distance capped at a prespecified value; all paths longer than that will be included in the matrix as having the maximum path length. Algorithm 1 shows the procedure for generating the distance matrix from the adjacency matrix for a given molecular graph.

Algorithm 1 Relative Position Matrix Creation

```
function CREATE_RPM(ADJ_MAT, MAX_PATH_LENGTH)

    fixed = n × n identity matrix
    D = n × n matrix in which all entries are equal to max_path_length.
    mat = adj_mat
    for i in range(max_path_length) do:
        condition_1 = indices where mat != 0
                ▷ there exists a path of distance ≤ i between these pairs of nodes
        condition_2 = indices where fixed = 0
                                ▷ haven't yet found a path between these nodes
        D[condition_1 ∩ condition_2] = i+1
        fixed[condition_1] = 1
        mat = matrix_multiply(mat, adj_mat)
    set diagonal elements of D to 0          ▷ distance from a node to itself is 0
    return D
```

When incorporating relative positions into the attention architecture, we can modify the equations accordingly. As done in [23], we create embedding vectors for each distance in the range (1, max_length). For each pair of nodes i and j, we have the position vector

$$a_{ij} = w_{clip(i,j)} \tag{4.4}$$

where

$$clip(i, j) = min(abs(dist(i - j)), k) \tag{4.5}$$

for maximum path length k.

Thus we have our new attention output with positional embeddings for node i:

$$\text{attention\_output}_i = \sum_j \text{softmax}\left(\frac{(z_q)(z_k + a_{ij})^T}{\sqrt{d_z}}\right)(x_j W_v + a_{ij}) \tag{4.6}$$

4.4.4 3D Distance Embeddings

In addition to relative position embeddings, we incorporated 3-dimensional distance embeddings into our network. Relative position embeddings used solely the graph representation of the molecule. It uses the minimum path length between any two nodes as the relative distance. In contrast, 3D embeddings take advantage of the

molecule's 3D conformation in space and the interactions that relate to this configuration. Figure 4-5 shows two configurations of the molecule triethylamine. For the 3D distance embeddings, our goal was to compute pairwise distances between all atoms and then use this distance information as part of the self-attention layer.

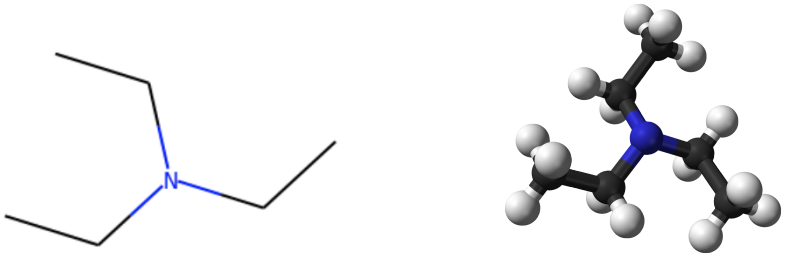

Figure 4-5: The 2D (left) and 3D (right) configurations of triethylamine. The 3D representation captures distances and orientations better than the 2D representation does. 3D diagram by Ben Mills - Own work, Public Domain, `https://commons.wikimedia.org/w/index.php?curid=3177079`

We used an RDKit method to precompute the 3D distance matrix for each SMILES string in our datasets. This method returns the floating point distances between all pairs of atoms in the molecule. Since we need to be able to have a finite set of distances in order to be able to use an embedding matrix, we discretized these distances by rounding to the nearest 0.5. As shown in figure 4-6 and 4-7, the distribution of distances in the BBBP dataset ranges from 0 (distance to self) to around 37 and the distribution of distances in the ClinTox dataset ranges from 0 to around 33.5. (For clarity, the graphs shown here do not include the 0 self distances, but rather cover the interval (0, max_dist], since there are far more self distances than any other value present in the set of distances.) Then, since the RDKit method only included the distances between atoms, we put a filler value in to represent distances between any component and a bond. Now, with this new representation of distances, we were able to pass this into our network as the distance matrix in place of the relative position matrix.

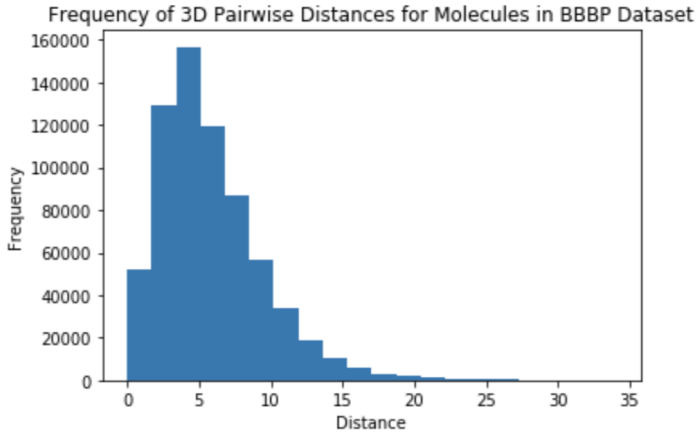

Figure 4-6: Histogram of pairwise distances between atoms in the BBBP dataset.

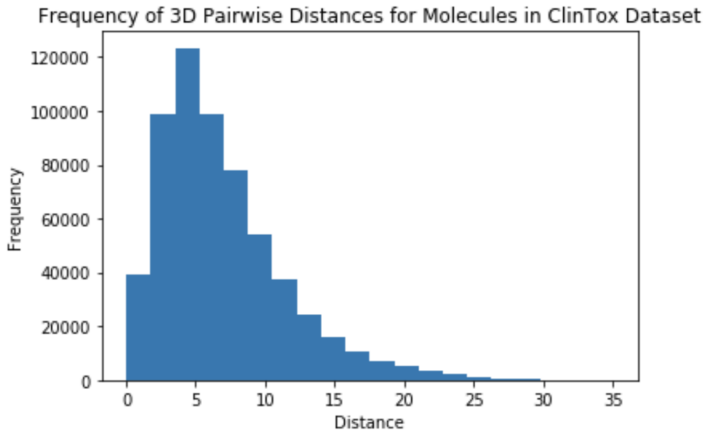

Figure 4-7: Histogram of pairwise distances between atoms in the ClinTox dataset.

4.4.5 Full Network Architecture

We combine the elements examined in this section in order to produce a final network architecture which we use to evaluate test performance and report final results. We organize the network using blocks, in which each block contains an attention layer, a node convolution layer, a ReLU activation layer, and a dropout layer. Each block also contains a residual connection which adds the original signal to the processed signal to produce the output. The architecture for a single block is shown in figure 4-8.

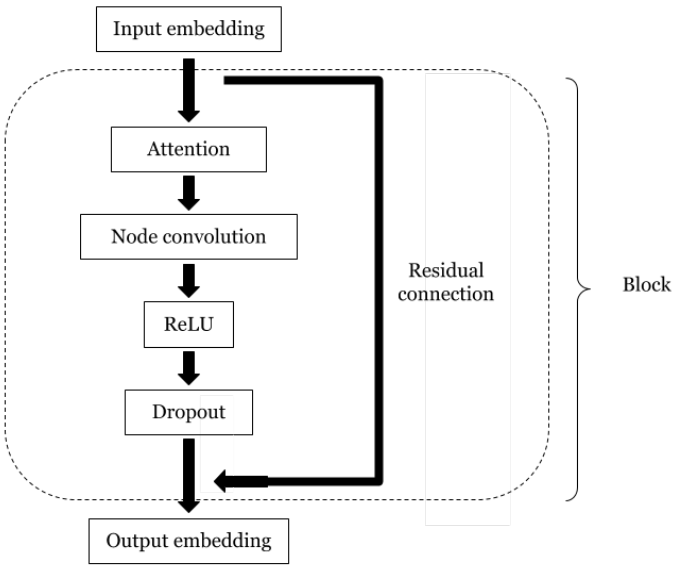

Figure 4-8: Architecture of a single block. Layers and residual connections are labeled to illustrate the flow of processing through the drug embedder.

4.5 Prediction

As a component of prior work, collaborators had developed a pipeline for molecular property prediction. This model takes in a dataset and a PyTorch model architecture, performs training, and evaluates the performance on several metrics. I was able to use this pipeline to test several different architectures and perform hyperparameter tuning and evaluation. Since creation of this pipeline was not of central focus to the present work, we will not describe it in detail. Instead, we will cover the method of network optimization.

4.5.1 Hyperparameters

The hyperparameters we tuned were: number of epochs, batch size, dimension of embedding vector, initial learning rate (which subsequently decays over time), number of blocks, dropout, and type of pooling.

- *Number of epochs* controlled how long the network had to train.
- *Batch size* determined how many samples were processed at once.

- *Dimension of the embedding vector* influences how much information the vector can capture: in general, larger vectors should be more expressive than smaller vectors.

- The *learning rate* ensured that the network trained quickly enough, but not so fast to lead to vanishing or exploding gradients.

- The *number of blocks* determined the layers that would make up the network configuration. We used a standard block that consisted of an attention layer, then a node convolution layer, then a ReLU activation layer, and then finally a dropout layer. When the number of blocks was one, the entire network consisted of the architecture specified in the block structure. If the number of blocks was larger, that structure was repeated that many times to comprise the network.

- The *dropout* hyperparameter was the standard usage where we varied the dropout percentage in order to prevent overfitting to the training data.

- We chose from among three types of *pooling*: max pooling, sum pooling, and multi pooling. As described earlier, pooling is a way of combining information and reducing the size of information passing through the network. Max pooling takes the largest value from among its receptive field. Sum pooling adds together the values in its receptive field. Our multi-pooling module combined a mean pooler and a max pooler and outputted both results so as to incorporate two different types of pooling.

In order to optimize these hyperparameters, we used random search. This consisted of randomly choosing a set of hyperparameters and running an experiment with those settings and then evaluating both training and validation set performance. At the end of a certain number of experiments, we chose the hyperparameters that performed best on the validation data. Algorithm 2 shows the process of runnning random hyperparameter search. This random hyperparameter tuning was also supplemented with manual tuning. As input to our hyperparameter search, we specified some hyperparameters with options of particular values and some with distributions.

We describe the hyperparameter space in table 4.1. Distributions are all from the SciPy Stats package.

Algorithm 2 Random Hyperparameter Search

```
function RANDOM_HYPERPARAMETER_SEARCH

    for i in range(num_iterations) do
        for each hyperparameter do
            if hyperparameter is pooling then
                pooler_type = random.choice(hyperparameters[pooling])
            else
                distribution = random.choice(hyperparameters[hyperparameter])
                hyperparameter_value = distribution.sample()
        model = new_model(hyperparameter_values)
        model.train()
        model.evaluate()
```

Hyperparameter	**Values**
Number of epochs	[poisson(30), poisson(60)]
Batch size	[poisson(10), poisson(30)]
Embedding vector dimension	[poisson(10), poisson(25), poisson(40)]
Initial learning rate	[lognorm(scale=1e-4, s=1)]
Number of blocks	[poisson(2)]
Dropout	[uniform(0, 0.25)]
Pooling	[Multi_Pooler, Sum_Pooler, Max_Pooler]

Table 4.1: Hyperparameters

Chapter 5

Results

5.1 Comparison

In order to evaluate our architecture, we focused on two datasets: BBBP and ClinTox. These two datasets were chosen because of their size and because they illustrate different properties of the molecules. Due to computational constraints, we demonstrate the performance of our model on these datasets and plan to fully profile other datasets in future work. We use Chemprop's command line interface to specify exact splits and obtain comparable results from their architecture.

In our evaluation of each dataset, we focus on three of the splits given by Chemprop: splits 0, 1, and 2. We perform hyperparameter tuning on the training set and choose the hyperparameters that give the best validation performance. Then, in order to obtain test performance numbers, we train on the combined training and validation sets and evaluate on the held out test set.

5.2 Evaluation Metrics

We use AUROC, or area under the receiver-operating characteristic, as our evaluation metric [9]. The receiver-operating characteristic plots the true positive rate against the false positive rate for various thresholds in a binary classification problem. At each threshold value, a single point is plotted (i.e. there is a single true positive rate

and a single false positive rate for performing classification using that threshold). For a very low threshold, all inputs will be classified as "positive", so both true positive and false positive will be at 1. For a very high threshold, no inputs will be classified as "positive" and so the true positive and false positive will be 0. A straight line going from (0, 0) to (1, 1) indicates the performance of a random classifier. The area under this line is 0.5, which indicates the essentially random performance and lack of predictive value. Conversely, a binary classifier that has positive predictive value lies above the straight line and therefore has an area of greater than 0.5.

We choose AUROC since our baseline comparison models use this metric and so it is easier to compare performance. This choice makes sense for imbalanced datasets like those in MoleculeNet. A relatively high accuracy can often be achieved by chance on a skewed dataset, but the AUROC will still be 0.5 for a random classifier.

5.3 Ablation Studies

We perform a series of ablation studies to demonstrate how various components of the network contribute to final performance. The components we wanted to analyze were the following:

- Presence of residual connections
- Presence of a convolution layer
- Presence of an attention layer
  - Attention with relative position embeddings
  - Attention with 3D position embeddings
  - Attention with no position embeddings

The description of each of our modified architectures is shown in figure 5.1. Note that the top two rows, '3D position embeddings' and 'Relative position embeddings' are two different types of distance embeddings. We hypothesized that these two architectures, which incorporate convolution, attention, and distance representations,

would perform well. In order to evaluate the relative contribution of these elements, we tested several other modified architectures as well. 'No residual connection' is the same as the '3D position embeddings' architecture but without the residual connection. 'No position embeddings' architecture includes self-attention but does not use either 3D or relative distance. 'Self attention only' and 'Node convolution only' are the most bare-bones architectures as they include either convolution or attention, but not both, while all the other architectures include both types of layers. These names are used throughout the results tables for continuity.

Modified architecture	**Conv.**	**Residual**	**Attn.**	**RP**	**3D**
3D position embeddings	X	X	X		X
Relative position embeddings	X	X	X	X	
No residual connection	X		X		X
No position embeddings	X	X	X		
Node convolution only	X	X			
Self attention only		X	X		

Table 5.1: Features present in each ablation study. We indicate the presence of a convolutional layer, residual connections, an attention layer, and, if there is an attention layer, presence of relative position embeddings or 3D position embeddings.

The performance of each of these modified architectures is shown on the BBBP and ClinTox datasets for validation and test sets in tables 5.2, 5.3, 5.4, and 5.5. Chemprop performance numbers are included as a reference.

We observed that ClinTox results were more variable than BBBP results and thus some of our results may have been due to random initial seeds rather than a meaningful difference in performance. In order to counteract this, each of the ClinTox validation and test numbers reported in 5.4 and 5.5 is an average over three different seeds.

Bolded results in the validation tables (5.2 and 5.4) show the best performing architecture in each split. Bolded results in the test tables (5.3 and 5.5) show the best performing architectures on each split; note that this is different than the test performance obtained from the best performing model on validation data.

Modified architecture	Split 0	Split 1	Split 2	Avg.
3D position embeddings	0.951	**0.963**	0.894	0.936
Relative position embeddings	0.945	0.941	**0.952**	**0.946**
No residual connection	**0.957**	0.939	0.929	0.942
No position embeddings	0.942	0.944	0.902	0.929
Node convolution only	0.941	0.947	0.926	0.938
Self attention only	0.854	0.851	0.776	0.827

Table 5.2: BBBP validation performance

Modified architecture	Split 0	Split 1	Split 2	Avg.
3D position embeddings	**0.937**	0.949	0.906	0.931
Relative position embeddings	0.905	0.943	**0.948**	0.932
No residual connection	0.936	**0.953**	0.857	0.915
No position embeddings	0.912	0.951	0.937	**0.933**
Node convolution only	0.887	0.946	0.942	0.925
Self attention only	0.862	0.782	0.849	0.831
Chemprop	0.931	0.893	0.954	0.926

Table 5.3: BBBP test performance

Modified architecture	Split 0	Split 1	Split 2	Avg.
3D position embeddings	0.630	0.569	0.785	0.661
Relative position embeddings	0.713	**0.809**	0.791	**0.771**
No residual connection	0.638	0.762	**0.817**	0.739
No position embeddings	0.727	0.566	0.705	0.666
Node convolution only	**0.728**	0.619	0.660	0.669
Self attention only	0.625	0.665	0.645	0.645

Table 5.4: ClinTox validation performance

Modified architecture	Split 0	Split 1	Split 2	Avg.
3D position embeddings	0.610	0.467	0.694	0.590
Relative position embeddings	**0.807**	0.701	**0.716**	**0.743**
No residual connection	0.620	0.681	0.707	0.669
No position embeddings	0.698	0.626	0.522	0.616
Node convolution only	0.565	0.683	0.554	0.601
Self attention only	0.552	**0.709**	0.443	0.568
Chemprop	0.902	0.788	0.916	0.869

Table 5.5: ClinTox test performance

5.4 Main Results

As our main result, we choose the best performing architecture *and* set of hyperparameters from each split, and then average those performance numbers over all three splits. This analysis treats the architecture as a sort of hyperparameter itself. Table 5.6 shows this best test performance on BBBP and on ClinTox. Architectures were chosen based on validation performance but test numbers are reported here. Chemprop's results are reported on identical splits and using their command line feature.

Task	**Our Test AUROC**	**Chemprop Test AUROC**
BBBP	0.944	0.926
ClinTox	0.658	0.869

Table 5.6: Best performance on BBBP and Clintox

Chapter 6

Discussion

In order to maintain fairness across all modified architectures, we ran a hyperparameter tuning script that performed 60 experiments, each with parameters randomly drawn from the distributions described in table 4.1. We expected that the network architecture as shown in figure 4-8 with 3D positional embeddings would perform the best. This architecture uses node convolution to take advantage of local patterns and self-attention to take advantage of global similarity. The residual connection allows the original signal to be preserved. Finally, 3D positional embeddings should incorporate information about the positions of atoms and bonds within a molecule.

It is clear that BBBP performance was much higher than that of ClinTox. While our best test performance on BBBP is close to or outperforms Chemprop, ClinTox significantly underperforms. This may be due to several reasons, but the most likely is that our model was somehow less suited to the ClinTox task than to BBBP. Perhaps we were searching a poor subset of hyperparameter space or perhaps the task is simply harder and our model is unable to account for the extra difficulty in prediction. BBBP did have more samples than ClinTox (2039 vs. 1478), but we doubt that dataset size alone would have such a significant impact on performance.

6.1 BBBP

The BBBP results closely mirrored our hypothesis about how various components of the network should contribute to final performance. The best performing models on the test set were the architectures with 3D position embeddings, relative position embeddings, or 3D embeddings with no residual connection. All three of these architectures use an attention layer with position embeddings and a node convolution layer. The high performance of the model with no residual connection may suggest that the residual connections do not have a significant impact on the performance of the model. An interesting result when looking at the average BBBP test performance is that the architectures with no position embeddings, with 3D position embeddings, and with relative position embeddings performed almost identically. This result may suggest that position embeddings are not actually as beneficial as it may seem by looking at each of the splits individually.

Finally, networks with only convolution layers or only attention layers underperformed the architectures with both attention and convolution layers, thus indicating that the combination of these two layers leads to improved performance on the BBBP prediction task.

Tables 6.1, 6.2, and 6.3 compare the optimal hyperparameter values found for each of the modified architectures in the ablation study. Many of the optimal architectures only have one block, which means that there is a limited amount of information that the network can process. This means that there is only one attention and/or convolution layer so the network processes relatively few relationships within the molecule as a whole. We can consider general trends among architectures that performed well but must be cautious in drawing strong conclusions due to the limited size of the random search performed and, of course, the differences between the architectures themselves.

Hyperparam	3D position	Relative position	No residual	No position	Conv. only	Attn. only
Num. epochs	28	55	33	60	58	64
Batch size	24	20	60	7	5	44
Embedding vector dim.	33	24	58	16	47	43
Init. learning rate	0.00173	0.00040	0.00040	0.00163	0.00052	0.00590
Num. blocks	1	1	1	1	2	2
Dropout	0.20272	0.19222	0.08296	0.11044	0.06367	0.15835
Pooling	Multi	Max	Max	Sum	Max	Multi

Table 6.1: Comparison of BBBP hyperparameters on split 0.

Hyperparam	3D position	Relative position	No residual	No position	Conv. only	Attn. only
Num. epochs	72	41	55	74	55	26
Batch size	42	72	64	14	54	24
Embedding vector dim.	47	32	37	47	38	11
Init. learning rate	0.00043	0.00105	0.00033	0.00088	0.00101	0.00073
Num. blocks	1	1	1	1	4	1
Dropout	0.21445	0.22723	0.12355	0.23914	0.06032	0.20159
Pooling	Multi	Max	Sum	Multi	Multi	Multi

Table 6.2: Comparison of BBBP hyperparameters on split 1.

Hyperparam	3D position	Relative position	No residual	No position	Conv. only	Attn. only
Num. epochs	65	55	56	29	54	59
Batch size	62	26	34	10	22	58
Embedding vector dim.	33	38	10	27	33	41
Init. learning rate	0.00052	0.00280	0.00051	0.00159	0.00078	0.00033
Num. blocks	2	1	1	1	2	1
Dropout	0.00821	0.13788	0.05492	0.05440	0.21646	0.07857
Pooling	Multi	Multi	Max	Multi	Sum	Multi

Table 6.3: Comparison of BBBP hyperparameters on split 2.

6.2 ClinTox

Although less in line with our hypothesis about the performance of a network with 3D embeddings, the ClinTox dataset still yielded interesting results. While we were able to match Chemprop's performance on the BBBP dataset, we were unable to do so with ClinTox. The highest performing architecture for ClinTox only achieved an AUROC of 0.807, while Chemprop achieved a test AUROC of 0.9. In addition, 3D position embeddings performed quite poorly, while relative position embeddings performed well. The architecture with convolution and self-attention but no position embeddings performed better than either convolution or self attention alone, in line with our predictions.

Overall we can see that the ClinTox results are much more varied than the BBBP results. Several of the AUROC values are close to 0.5, indicating that the prediction is no better than random. Validation results range from 0.566 to 0.817 and test results range from 0.467 to 0.807. In general, the validation and test performance is not as well correlated as those of BBBP. While generally it appears that architectures with only convolution or self-attention perform worse than architectures with both types of layers, this is not always true. 'Node convolution only' performed the best on split 0, and 'Self attention only' performed the best on split 1.

Tables 6.4, 6.5, and 6.6 show the optimal hyperparameter values obtained for each modified architecture. Like the high variation in test performance, the hyperparameters seem to show no particularly clear trends.

Although future work would include further examination of the ClinTox and other datasets, for now we conclude that our methodology worked well on the BBBP dataset but requires further study for improved performance on ClinTox.

Hyperparam	3D position	Relative position	No residual	No position	Conv. only	Attn. only
Num. epochs	33	28	31	48	69	66
Batch size	28	26	16	10	7	60
Embedding vector dim.	11	41	13	26	34	11
Init. learning rate	0.00036	0.00480	0.00083	0.00656	0.00081	0.00073
Num. blocks	1	1	1	1	3	1
Dropout	0.16297	0.22523	0.18397	0.02709	0.23228	0.00434
Pooling	Sum	Max	Max	Multi	Sum	Multi

Table 6.4: Comparison of ClinTox hyperparameters on split 0

Hyperparam	3D position	Relative position	No residual	No position	Conv. only	Attn. only
Num. epochs	64	25	62	30	45	61
Batch size	58	10	10	54	10	30
Embedding vector dim.	7	32	43	35	47	45
Init. learning rate	0.00006	0.00017	0.00071	0.00100	0.00055	0.00113
Num. blocks	1	1	2	2	3	3
Dropout	0.18010	0.14443	0.13969	0.11906	0.24887	0.01939
Pooling	Max	Sum	Max	Max	Max	Max

Table 6.5: Comparison of ClinTox hyperparameters on split 1.

Hyperparam	3D position	Relative position	No residual	No position	Conv. only	Attn. only
Num. epochs	36	72	61	67	30	30
Batch size	28	26	16	24	6	22
Embedding vector dim.	35	38	39	42	39	8
Init. learning rate	0.00226	0.00095	0.00100	0.00048	0.00089	0.00174
Num. blocks	1	1	1	1	2	2
Dropout	0.10574	0.19516	0.11803	0.04686	0.17636	0.00103
Pooling	Multi	Max	Sum	Max	Multi	Multi

Table 6.6: Comparison of ClinTox hyperparameters on split 2.

6.3 Limitations

There are several important limitations to the work presented here.

The first limitation relates to the graph convolution operation. We use four functions to perform operations on each of a node's (up to four) neighbors. However, the ordering of these nodes is consistently given in an arbitrary order as obtained through RDKit. This behavior is not ideal, as in reality the ordering of the nodes should not matter, since they are not really ordered in the molecule itself. Thus, in future work we would want to somehow shuffle the ordering of the nodes or ensure that the functions are not somehow using this arbitrary ordering.

Another limitation is due to the scope of this work. We study only two datasets from MoleculeNet and obtain results for a subset of the splits provided by Chemprop. In future work, we would want to run experiments on all splits and then obtain average results with confidence intervals, as given in Chemprop.

Other works, including MoleculeNet, have found a multi-task approach to be highly beneficial in performing classification [30]. This involves predicting multiple tasks at once, which allows for potentially overlapping information to be shared. We focused on single-task classification in this work but hope to incorporate multi-task as well in future work.

Finally, in performing hyperparameter tuning, our random search space was initially limited and random search itself inherently may not find the best possible model. Thus close comparisons between different architectures can not be assumed to be statistically significantly different.

Chapter 7

Conclusion and Future Work

In this work, we introduced a novel deep learning architecture that performs molecular property prediction. We combine graph convolution with self-attention and incorporate 3D distance embeddings. We demonstrate that this architecture is able to perform molecular property prediction as well as state of the art on the BBBP dataset. By performing ablation studies, we analyzed the contribution of each element of the network. We found results that indicate that a combination of graph convolutions and self-attention performs better than either type of network alone.

In the future, we plan to profile more of the MoleculeNet datasets to see how much our results generalize. While BBBP results were consistent with our hypothesis, ClinTox results were not as performant. We would like to further investigate possible reasons for this discrepancy.

Deep learning holds great potential for applications in the pharmaceutical industry. By successfully predicting molecular properties with our network, we hope to contribute to the goal of improving the cost and efficiency of drug discovery.

Bibliography

[1] RDKit: Open-source cheminformatics. http://www.rdkit.org.

[2] Ossama Abdel-Hamid, Abdel Rahman Mohamed, Hui Jiang, Li Deng, Gerald Penn, and Dong Yu. Convolutional neural networks for speech recognition. *IEEE Transactions on Audio, Speech and Language Processing*, 22(10):1533–1545, 10 2014.

[3] Irwan Bello, Barret Zoph, Ashish Vaswani, Jonathon Shlens, and Quoc V. Le. Attention Augmented Convolutional Networks. 4 2019.

[4] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral Networks and Locally Connected Networks on Graphs. 12 2013.

[5] Michal Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. 2016.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. 10 2018.

[7] Joseph A. DiMasi, Henry G. Grabowski, and Ronald W. Hansen. Innovation in the pharmaceutical industry: New estimates of R&D costs. *Journal of Health Economics*, 47:20–33, 5 2016.

[8] David Duvenaud, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gómez-Bombarelli, Timothy Hirzel, Aln Aspuru-Guzik, and Ryan P. Adams. Convolutional Networks on Graphs for Learning Molecular Fingerprints. 9 2015.

[9] J A Hanley and B J McNeil. The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology*, 143(1):29–36, 4 1982.

[10] Hisaki Ikebata, Kenta Hongo, Tetsu Isomura, Ryo Maezono, and Ryo Yoshida. Bayesian molecular design with a chemical language model. *Journal of Computer-Aided Molecular Design*, 31(4):379–391, 4 2017.

[11] Richard M Karp. Reducibility among combinatorial problems. *https://people.eecs.berkeley.edu/ luca/cs172/karp.pdf*, 1972.

[12] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale Video Classification with Convolutional Neural Networks. Technical report.

[13] Steven Kearnes, Kevin McCloskey, Marc Berndl, Vijay Pande, and Patrick Riley. Molecular graph convolutions: moving beyond fingerprints. *Journal of Computer-Aided Molecular Design*, 30(8):595–608, 8 2016.

[14] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[15] Peter Kolb, Rafaela S. Ferreira, John J. Irwin, and Brian K. Shoichet. Docking and chemoinformatic screens for new ligands and targets, 8 2009.

[16] A Krizhevsky, I Sutskever, and GE Hinton. Imagenet classification with deep convolutional neural networks. pages 1097–1105, 2012.

[17] Yann LeCun, Lon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2323, 1998.

[18] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A Robustly Optimized BERT Pretraining Approach. 7 2019.

[19] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality. 10 2013.

[20] Bruno J. Neves, Rodolpho C. Braga, Cleber C. Melo-Filho, Jos Tefilo Moreira-Filho, Eugene N. Muratov, and Carolina Horta Andrade. QSAR-based virtual screening: Advances and applications in drug discovery, 11 2018.

[21] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Stroudsburg, PA, USA, 2014. Association for Computational Linguistics.

[22] Marwin H.S. Segler, Thierry Kogej, Christian Tyrchan, and Mark P. Waller. Generating focused molecule libraries for drug discovery with recurrent neural networks. *ACS Central Science*, 4(1):120–131, 1 2018.

[23] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-Attention with Relative Position Representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 464–468, Stroudsburg, PA, USA, 2018. Association for Computational Linguistics.

[24] Fujian Tan, Ruizhi Yang, Xiaoxue Xu, Xiujie Chen, Yunfeng Wang, Hongzhe Ma, Xiangqiong Liu, Xin Wu, Yuelong Chen, Lei Liu, and Xiaodong Jia. Drug repositioning by applying 'expression profiles' generated by integrating chemical structure similarity and gene semantic similarity. *Molecular BioSystems*, 2014.

[25] Thomas Unterthiner, Andreas Mayr, Gnter Klambauer, and Sepp Hochreiter. Toxicity Prediction using Deep Learning. 3 2015.

[26] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. 6 2017.

[27] Yongcui Wang, Shilong Chen, Naiyang Deng, and Yong Wang. Drug repositioning by kernel-based integration of molecular structure, molecular activity, and phenotype data. *PLoS ONE*, 2013.

[28] Zichen Wang, Neil R. Clark, and Avi Ma'ayan. Drug-induced adverse events prediction with the LINCS L1000 data. *Bioinformatics*, 32(15):2338–2345, 8 2016.

[29] Robin Winter, Floriane Montanari, Frank Noé, and Djork Arn Clevert. Learning continuous and data-driven molecular descriptors by translating equivalent chemical representations. *Chemical Science*, 10(6):1692–1701, 2019.

[30] Zhenqin Wu, Bharath Ramsundar, Evan N. Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S. Pappu, Karl Leswing, and Vijay Pande. MoleculeNet: A benchmark for molecular machine learning. *Chemical Science*, 9(2):513–530, 2018.

[31] Kevin Yang, Kyle Swanson, Wengong Jin, Connor Coley, Philipp Eiden, Hua Gao, Angel Guzman-Perez, Timothy Hopper, Brian Kelley, Miriam Mathea, Andrew Palmer, Volker Settels, Tommi Jaakkola, Klavs Jensen, and Regina Barzilay. Are Learned Molecular Representations Ready For Prime Time? 4 2019.

[32] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. XLNet: Generalized Autoregressive Pretraining for Language Understanding. 6 2019.

[33] Zhengyan Zhang, Xu Han, Zhiyuan Liu, Xin Jiang, Maosong Sun, and Qun Liu. ERNIE: Enhanced Language Representation with Informative Entities. 5 2019.