Predicting Molecular Properties using Graph Neural Networks

Nithin Varadharajan
Luddy School of Informatics, Computing, and Engineering
Indiana University
Bloomington, IN 47408, USA
nvaradha@iu.edu

Sathish Soundararajan
Luddy School of Informatics, Computing, and Engineering
Indiana University
Bloomington, IN 47408, USA
satsoun@iu.edu

Ramki Ramamurthy
Luddy School of Informatics, Computing, and Engineering
Indiana University
Bloomington, IN 47401, USA
ramrama@iu.edu

***Abstract* - Graph neural networks, machine learning on graphs, has been one of the efficient ways to improve the predictive accuracy during the study of molecular properties which is the fundamental study in pharmaceutical industries especially in the drug development. The development of drugs is a rigorous and time-consuming process that involves the study of five molecular properties: Absorption, Distribution, Metabolism, Excretion, and Toxicity. In this project, we performed a comparative study to study the prediction of the three of the molecular properties: Absorption, Toxicity, and Solubility from the datasets TOX21, BBBP, and ESOL using several graph neural networks such as Message Passing Neural Network (MPNN), Graph Convolution Network, GraphSAGE and Graph Attention network. In addition to the GNN models, we also developed a MOL2VEC embedding and performed a downstream RandomForestClassifier model to compare the effectiveness. The results indicate that the graph neural network: GraphSAGE performed better relatively compared to other GNN and shallow embedding methods on average. In addition to predicting the molecular property, we visualized the important substructures that impact the prediction.**

Index Terms* - *Molecular property prediction, graph neural network: MPNN, GCN, GraphSAGE, GAT, MOL2VEC

I. Introduction

In today's world in which it is still not safe to declare that the pandemic has ended, the need and demand for effective and timely drug development is on the rise. A typical drug development process focuses on the discovery, design, identification, isolation, development of new molecular agents that could potentially cure a disease or treat a condition. In the United States, on an average it could take about a decade for a drug to be tested, approved and available for the general public with a rate of approval for the drug being at 12%. To generalize, it could require huge investments in several billions for a drug to be tested and approved. A significant part of the investment essentially goes down to failures in the early stages of the process where only a few molecules will receive an approval. This cost of failures can be brought down extensively by studying the molecular properties and filtering it in the initial step which is the essence of this project: to study the molecular properties.

With the advent of machine learning and with the encoded representation of molecular information in the form of a fingerprint, the prediction of molecular predictions was made less time consuming and with better accuracy. A molecule can also be represented in the form of a graph with atoms representing a node and the bonds representing the edges. Graph neural networks is the most widely used deep learning method for tasks that involve graph data. With Graph Neural Networks which is extensively used over the last few years in molecular pharmacology, this representation of molecules in the form of a graph can be exploited to predict the molecular properties. Graph neural networks can learn the representations in an automated way and in turn get rid of the convoluted feature engineering process.

II. Method

A. *MOL2VEC*

In NLP terms, each molecule forms a sentence, and each substructure forms a word in Mol2Vec. By applying Word2Vec on the set of molecules, we obtain a high dimensional embedding of substructures where the vectors of the chemically related substructures occupy the same part of vector space in a closer proximity to each other. Mol2Vec is an unsupervised method that is trained on the input unlabeled data to obtain feature vectors of substructures which could be summed up to obtain molecule vectors.

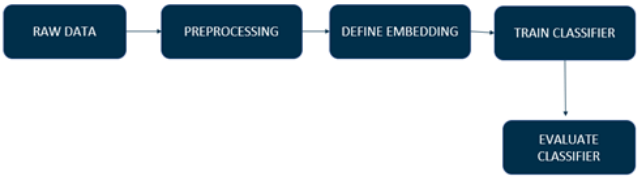

Fig. 1 MOL2VEC Data Flow Chart

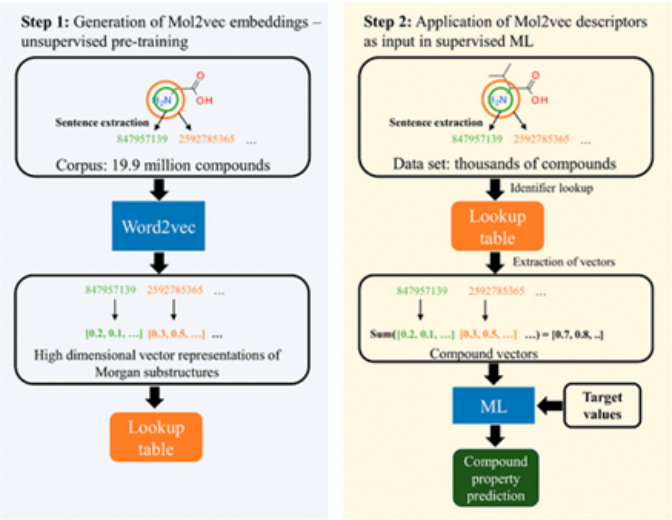

Fig. 2 MOL2VEC Flow Chart for Unsupervised and Supervised Methods

In the preprocessing phase for Mol2Vec approach, we extract the atom and bond features which represent the node and edge features respectively. Once extracted, we apply the Word2Vec on the molecular data represented in the SMILES format to obtain a high dimensional Mol2Vec embedding. This is commonly referred to as the pre-training phase. For the prediction of the molecular properties, we developed a random forest classifier and neural network with 3 layers with 300, 20 and 12 or 1 in each of the layers (the output layer = 12 neurons for Tox21 dataset and 1 for BBBP dataset). The model was then tuned with different hyperparameter settings and the best model was used as the classifier. The results of the classifier will be discussed in the results section.

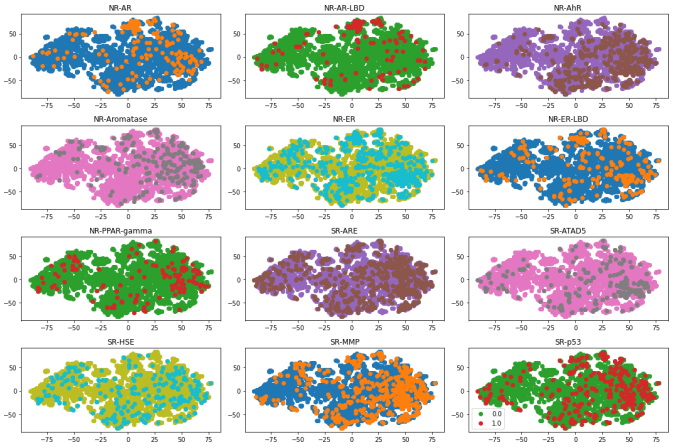

Fig. 3 Multi Label Classification of Embeddings obtained from Tox21 Dataset using Mol2Vec.

B. *GNN - MPNN*

Message Passing Neural Networks utilize the notion of messages in GNNs. A message m_ij can be sent across edges i and j and is computed using a message function f_e . f_e is generally a small MLP and takes into consideration both the nodes and edges features. In MPNN's, the forward pass has two phases, a message passing phase and a readout phase. The message passing phase runs for T time steps and is defined in terms of message functions M_t and vertex update functions U_t. For the update functions we have used the Gated Recurrent Unit as mentioned in the paper. During the message passing phase, hidden states Ht at each node in the graph are updated based on the aggregate neighborhood messages according to the given equation below:

$$m_v^{t+1} = \sum_{w \in N(v)} M_t(h_v^t, h_w^t, e_{vw})$$

$$h_v^{t+1} = U_t(h_v^t, m_v^{t+1})$$

Fig. 4 Message Passing aggregation and update equations

The readout phase computes a feature vector which is the D dimensional embedding vector for the whole graph using some readout function R according to the equation below:

$$\hat{y} = R(\{h_v^T \mid v \in G\}).$$

Fig. 5 Message Passing readout equation

This readout phase function is implemented by the Transformer encoder function to get the embedding vectors.

C. *GNN - GCN*

Generalizing well-established neural models like RNNs or CNNs to work on arbitrarily structured graphs is a challenging problem. In GCN, they took a somewhat similar approach and started from the framework of spectral graph convolutions and introduced simplifications by further approximations. For these models, the goal is then to learn a function of signals/features on a graph G {N, M} which takes as input:

- A feature description Xi for every node i, summarized in a N×D feature matrix X (N - number of nodes, D - number of input features).
- A representative description of the graph structure in matrix form; typically, in the form of an adjacency matrix A (or some function thereof)

And produces a node-level output Z (an N×F feature matrix, where F is the number of output features per node). Every neural network layer can then be written as a nonlinear function.

$$H^{(l+1)} = f(H^{(l)}, A)$$

Fig. 6 GCN Layer Equation

With H0=X and HL=Z (or z for graph-level outputs), L being the number of layers. The specific models then differ only in how f($\cdot,\cdot$) is chosen and parameterized.

We do this parameterization in GCN, by normalizing the Adjacency matrix multiplying with the inverse of node Degree matrix & node feature matrixes and approximate the result with a pooling operation.

$$f(H^{(l)}, A) = \sigma\left(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

Fig. 7 GCN Equation

With $\hat{A} = A + I$, where I is the identity matrix and D^ is the diagonal node degree matrix of A^. In the paper they relaxed some assumptions and took the graph convolution for 1 hop neighborhoods and stacked up multiple layers of GCNConv. In The Graph Convolution Layer (A^ * X) can be efficiently implemented as a product of a sparse matrix with a dense matrix.

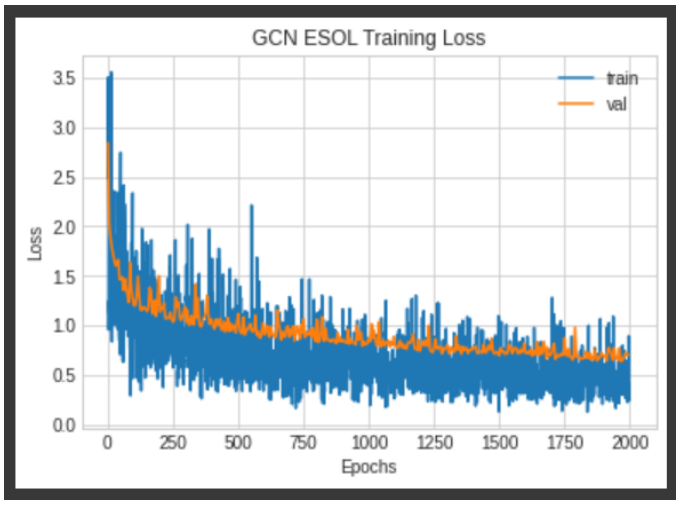

Fig.8 Training Loss in GCN for ESOL Dataset

C. *GNN - GraphSAGE*

We also looked at the GraphSAGE (Sample and Aggregate) variant of a graph neural network introduced by Hamilton et. al [4]. Like other GNN methods GraphSAGE is inductive, meaning that node features can be used to generate node embeddings for new data. Other methods like training low-dimensional embeddings are transductive and will not generalize. The main idea is that a function is learned that generates node embeddings by sampling and aggregating features from a node's neighborhood.

Fig. 9 Visual illustration of the GraphSAGE sample and aggregate approach

This way even if a new node unseen during training time appears in the graph, it can still be properly represented by its neighboring nodes.

Algorithm 1: GraphSAGE embedding generation (i.e., forward propagation) algorithm

Input : Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; depth K; weight matrices $\mathbf{W}^k, \forall k \in \{1, ..., K\}$; non-linearity σ; differentiable aggregator functions $\text{AGGREGATE}_k, \forall k \in \{1, ..., K\}$; neighborhood function $\mathcal{N} : v \rightarrow 2^{\mathcal{V}}$

Output : Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{V}$

$\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$;
for $k = 1...K$ **do**
  for $v \in \mathcal{V}$ **do**
    $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;
    $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$
  end
  $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$
end
$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$

Fig. 10 Pseudocode of the forward propagation GraphSAGE

In the above pseudocode for GraphSAGE, every iteration the current node is updated by the output of the neighborhood aggregation, the previous iterations vector value and the weight vector. The implementation we use uses a mean aggregator. This aggregator takes the average of the previous vector v and all the nodes in v's neighborhood. This is then multiplied by the weight of the vector passed through the activation function to update the current node.

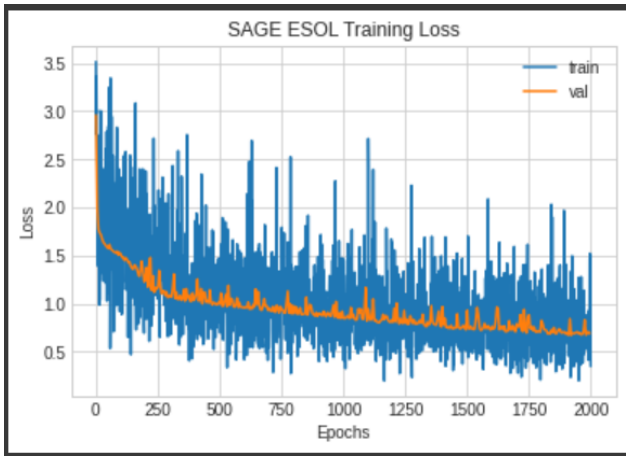

Fig.11 Training Loss in GraphSAGE for ESOL DataSet.

D. *GNN - GAT*

Another GNN model we investigated was graph attention networks (GATs). GATs enable in essence specifying different weights to different nodes in a neighborhood. GCNs and GraphSAGE use isotropic aggregation, meaning each neighbor contributes equally to the update function. GATs offer a potential increase in performance due to the learnable attention mechanism, which will weigh each neighbor's contribution to the update.

$$z_i^{(l)} = W^{(l)} h_i^{(l)}, \tag{1}$$

$$e_{ij}^{(l)} = \text{LeakyReLU}(\vec{a}^{(l)^T}(z_i^{(l)} || z_j^{(l)})), \tag{2}$$

$$\alpha_{ij}^{(l)} = \frac{\exp(e_{ij}^{(l)})}{\sum_{k \in \mathcal{N}(i)} \exp(e_{ik}^{(l)})}, \tag{3}$$

$$h_i^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij}^{(l)} z_j^{(l)}\right), \tag{4}$$

Fig. 12 Equations for GAT

The attention coefficient is computed based on node features, which are fed into an attention function. The pair-wise attention score is calculated by concatenating the embeddings of two nodes, taking the dot product of that embedding with a weight vector and finally passing the output through a LeakyReLU function. This can be seen in eq. (2) in the figure. Then attention scores are passed through a softmax to get a probability distribution. Now this score can be used to weight each neighbor's contribution to the current node's update.

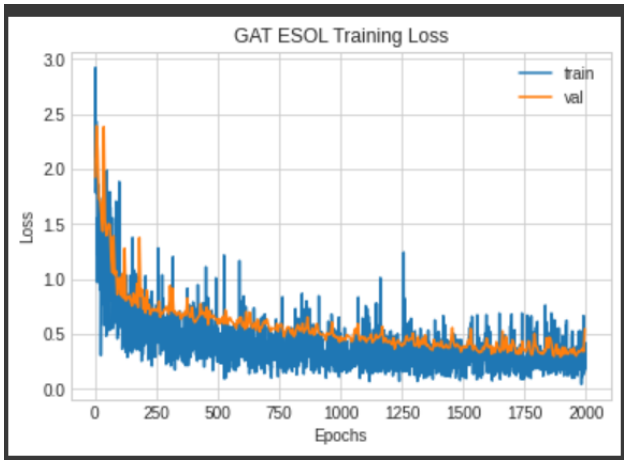

Fig.13 Training Loss in GAT for ESOL DataSet.

E. *GNNExplainer*

Recent research works have shown GNNs power for understanding graphs, but the way how and why GNN works remains a mystery for most people. In CNNs, we can extract activation of each layer to visualize the decisions of the network, but in GNN it's hard to get a meaningful explanation of what features the network has learnt. GNNExplainer creates a wrapper around the GNN Model and tries to reduce the redundancy information by creating a minimal subset of nodes and edge information and tries to visualize the importance of certain nodes and edge relations in the graph.

Basically, how gnn-explainer works is by generating a minimal graph that explains the decision for a node or a graph. The problem we have in hand is an optimization problem of minimizing the difference between the prediction

scores using the whole computation graph and the minimal subgraph.

$$\max_{G_S} MI\left(Y, (G_S, X_S)\right) = H(Y) - H(Y|G = G_S, X = X_S).$$

Fig. 14 Optimization equation for GNNExplainer

In the paper, this process is formulated as maximizing the mutual information (MI) between the minimal graph Gs and the computation graph G. the graph needs to be minimal, and this objective is maintained by adding a loss for the number of edges. Therefore, the loss for GNNExplainer is the combination of prediction loss and edge size loss.

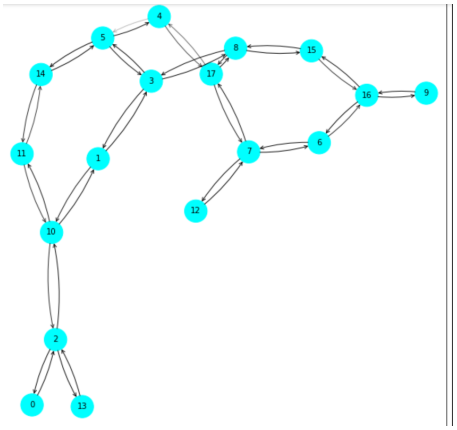

Fig 15: GNN-Explainer Visualization for Flucindol – explaining the importance of nodes and relational information.

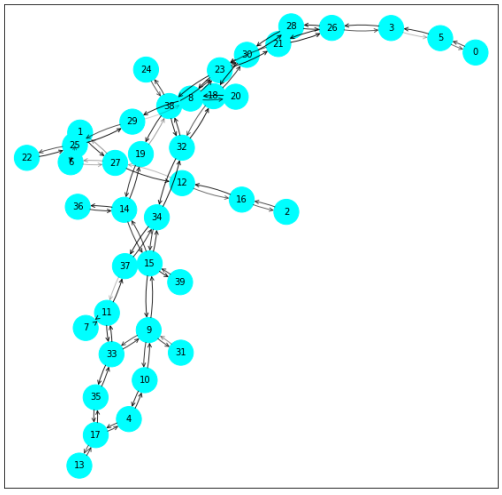

Fig 16: GNN-Explainer Visualization for Itrocinonide - determining importance of nodes and relational information.

III. Experimental Setup

To give an overview, our experiments consist of applying the GNN and embedding methods we have described to our chosen Molecule Net datasets, TOX21, BBBP, and ESOL. We have 5 methods in total, four GNN based (GCN, GraphSAGE, GAT, and MPNN) and one embedding based (Mol2Vec). The datasets are from Molecule Net [2], which is an open benchmark for testing machine learning methods in the chemistry domain. Molecule Net is also part of a larger open-source machine learning chemistry library called DeepChem. DeepChem contains different molecule featurization tools and model implementations for the datasets in Molecule Net. The version of DeepChem employed was version 2.6.1.

The experiments were run using Google Colab and ran using Pytorch with the GPU allocated by Colab. For us, this was usually a Nvidia Tesla K80 or a faster Nvidia Tesla P100. The experiments never used more than 10 GB of RAM. The largest TOX21 model took 40 minutes to train with the largest hyperparameters.

The BBBP (Binary labels of blood-brain barrier penetration(permeability)) dataset indicates whether a molecule can pass through the blood-brain barrier into the brain. A “0” is permeable, while a “1” represents not permeable. This is a classification problem.

The classes in most of these datasets are also imbalanced. For the BBBP dataset the non-permeable samples outnumber the permeable samples almost 4 to 1. This is something that needs to be considered when deciding on a metric for evaluation. The dataset has around ~2k samples. The ESOL dataset is water solubility data for common organic small molecules. It is a regression task; in that we would need to estimate the log solubility in moles per liter. This dataset is around ~3k samples. Finally, TOX21 is a multi-label classification problem that has toxicity measurements on 12 biological targets, including nuclear receptors and stress response pathways. There are many different toxicity effects such as carcinogenicity, respiratory toxicity, and corrosion. This dataset is much larger than the previous datasets at ~12k samples. This greatly increases the training time with a sufficiently complex model.

TOX21 had some missing values in certain columns. For the purposes of our experiment, we decide to impute the missing values with zero.

The Datasets themselves are usually imported as comma separated files that then need to be preprocessed before they can be used in the models. The molecules in the dataset are in the SMILES (Simplified Molecular Input Line Entry System) string format. The information on the molecule and its constituent atoms and properties are contained in the SMILES string. This data can then be parsed for each molecule to identify node features and edge features for each molecule. RDkit makes dealing with chemical molecules easy, as we can simply retrieve molecule properties from the molecule object. We are using the RDkit-pypi version (2022.3.2). So, a smile string can be converted into a graph form as an adjacency matrix, or it can be embedded in a molecular fingerprint or a word embedding.

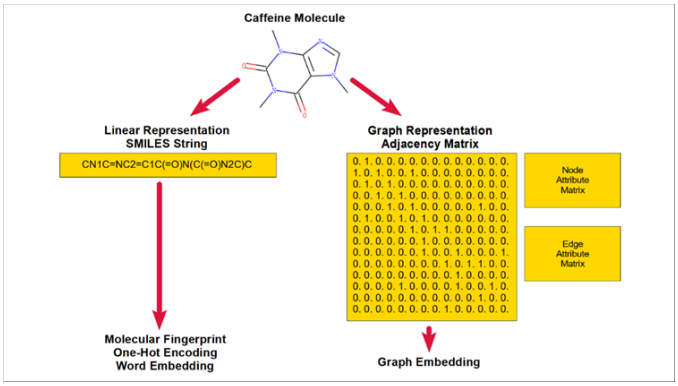

Fig. 17 Caffeine Molecule Data Representations

There are some molecules that will be difficult to featurize or they might throw an error. The MolGraphConvFeautrizer from DeepChem is used for most of our experiments. These problematic molecules need to be dropped from the dataset before the data can be built into a torch-geometric dataset. Torch-geometric is a library built on top of Pytorch for GNN applications, and it holds all our GNN implementations. We are using torch-geometric version 2.0.4, and Torch version 1.11.0.

Finally, when our dataset preparation is finished, we will have node features, edge features, edge index, and a target for each molecule in the dataset. For most of our experiments we perform an 80/10/10 train validation test random split.

For our GNN implementations we then use Pytorch to build the GNN architecture that will train on our dataset to generate an embedding that is used for our downstream prediction task. The architecture of our model is standard, with a variable number of GCN convolution blocks with a nonlinear activation function after each. The node features and edge index of each batch is passed into each convolution block. Followed by a global pooling layer, and finally some dense layers to perform the downstream task. All the Pytorch models in our experiments use one or two dropout layers before the prediction to help with the model generalization and to prevent overfitting. Adam is used as the optimizer for all the models, SGD (stochastic gradient descent) was found to have slightly worse performance. Rectified linear activation function is used as the non-linearity in most models, except for the TOX21 models as a tanh function was found to work better. These are the hyperparameters that are constant for all models.

The hyper parameters we chose to evaluate for the GNN models are learning rate, hidden layer dimension, and number of convolutional layers. The number of epochs the model ran for was determined by the dataset. The BBBP models ran for 200 epochs, the TOX21 models 100 epochs, and the ESOL models 2500 epochs. These values were found to allow the models to fit the data well. Binary cross entropy loss was used on all classification tasks and mean squared error loss was used for the regression task.

The MPNN model implementation is present in DeepChem and was used for our experiment. For the MPNN experiments we performed a 60/20/20 train validation test random split. The hyper parameters we focused on for the MPNN model were the learning rate, the size of the edge hidden features, and the number of message passing steps. The MPNN was run for 100 epochs each.

Our Mol2Vec implementation feeds the molecule vectors to the pre-initialized Word2vec model to generate embeddings. The model will give us 300-dimension embeddings which go through a classifier for our downstream task. The hyper parameters are the size of the hidden layer in the classifier. The activation function is a sigmoid, and it uses an Adam optimizer.Mol2Vec version used is 0.1, and it uses genism version 3.6.0.

IV. Results

Here is our result table where we have the F1(classification) or RMSE (regression) scores depending on the dataset and task for each of our 5 methods.

Models/F1 Score or RMSE	BBBP	Tox-21	ESOL
Mol2Vec	0.75	0.61	0.80
MPNN	0.8284	0.5086	0.7071
GCN	0.8003	0.578	0.69
GAT	0.868	0.5794	0.81
GraphSAGE	0.83	0.659	0.78

Table 1: Model Performance on all Datasets

From the table the GraphSAGE model performs the best for the Tox-21 data, while the MPNN performs the worst. For the BBBP dataset, the GAT model seemed to have worked the best. The second-best model for BBBP was the GraphSAGE model. This is inline to our expectations as a shallow embedding, should perform worse to supervised methods like GNNs. For ESOL we are looking at RMSE scores, so the lower the error the better the model performed on the test set. Here the supervised models outperformed the unsupervised methods.

Hyperparameters were selected for the GNN models by searching through a 3 x 3 search space for each of the 15 combinations of model and data.

GNN			
Learning_Rate	0.01	0.001	0.0001
Hidden Layer Dimension	64	128	256
Num Convolution Blocks	3	4	5
MPNN			
Learning_Rate	0.01	0.001	0.0001
Edge Hidden Feature Size	64	128	256
Num Message Passing Steps	3	4	5

Table 2: Model Hyperparameters for GNN an MPNN

The MPNN and the GNN models and the MOL2VEC had different hyper parameters to tune, so different types and ranges of parameters were selected depending on the task. For MOL2VEC we varied the classifier's hidden dimension size, learn rate, and optimizer. The data was tested on the validation data. Some parameters stayed constant for the GNN models like all models were run for either 100 or 200 epochs depending on the task. They all used Adam, and dropout. We tried to vary one parameter at a time, so usually the middle parameters were treated like a default kept constant while the other parameters varied. We got tables like those below.

GCN BBBP(F1_Score)

Learning_Rate	0.01	**0.001**	0.0001
	0.869	**0.926**	0.902
Hidden Layer Dimension	64	128	**256**
	0.919	0.933	**0.953**
Num Convolution Blocks	**3**	4	5
	0.969	0.907	0.923

Table 3: Hyperparameters chosen for GCN BBBP Dataset

V. Conclusion

In this project we set out to study the molecular properties of chemical compounds using benchmark datasets from Molecule Net. We infer that the prediction of molecular properties using GNNs is more effective compared to using a shallow embedding like Mol2Vec. We tuned hyperparameters for different GNNs and Mol2Vec and performed different downstream tasks like regression, multi-class, multi-label classifications for different molecules and analyzed their properties. We didn't have computational power for large chemical datasets. We would like to explore how pre-training with large chemical datasets affects our results. Future work would also include generative models that discover new and unique chemical compounds based on desirable properties.

VI. Contributions

We all worked on the project together and contributed equally. The literature review, dataset pre-processing, model implementations, designing test and acquiring results, and writing the presentation and final report were done collaboratively.

VII. References

[1]. Kipf, T. N. and Welling, M., "Semi-Supervised Classification with Graph Convolutional Networks", https://doi.org/10.48550/arXiv.1609.02907, 2016.

[2]. Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E., "Neural Message Passing for Quantum Chemistry", https://doi.org/10.48550/arXiv.1704.01212, 2017.

[3].Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio,Y.,"GraphAttentionNetworks",https://doi.org/10.48550/arXiv.1710.10903 , 2017.

[4]. Hamilton, W. L., Ying, R., and Leskovec, J., "Inductive Representation Learning on Large Graphs",https://doi.org/10.48550/arXiv.1706.02216, 2017.

[5].*J. Chem. Inf. Model.* 2018, 58, 1, 27−35 Publication Date:December 22, https://doi.org/10.1021/acs.jcim.7b00616,2017.

[6]. Lucic, A., ter Hoeve, M., Tolomei, G., de Rijke, M., and Silvestri, F., "CF-GNNExplainer: Counterfactual Explanations for Graph Neural Networks",https://doi.org/10.48550/arXiv.2102.03322, 2021.

[7]. Wu, Z., "MoleculeNet: A Benchmark for Molecular Machine Learning", https://doi.org/10.48550/arXiv.1703.00564 , 2017.

[8].Mayr A, Klambauer G, Unterthiner T and Hochreiter S (2016) DeepTox: Toxicity Prediction using Deep Learning. Front. Environ. Sci. 3:80. doi: 10.3389/fenvs.2015.00080.