

Modelling Drug-Drug Interaction Prediction using Graph Neural Networks

Submitted to Dr. Shiv Ram Dubey, Course Instructor (Deep Learning), IIIT Sricity

Abhishek Yadav
S20180010004
abhishek.y18@iiits.in

Mohit kumar
S20180010106
mohit.k18@iiits.in

Sathyanarayanan R
S20180010154
sathyanarayanan.r18@iiits.in

Shaikh Mohd. Fauz
S20180010160
mohdfauz.s18@iiits.in

S.No.	Group Member	Contribution
1	Abhishek Yadav (S20180010004)	<ul style="list-style-type: none">▪ Data loading▪ Initial features (random, one-hot, node2vec)▪ MLP Decoder
2	Mohit Kumar (S20180010106)	<ul style="list-style-type: none">▪ Training and Optimisation▪ Predictive Analysis
3	Sathyanarayanan R (S20180010154)	<ul style="list-style-type: none">▪ RGCN Encoder▪ MLP Decoder
4	Shaikh Mohd Fauz (S20180010160)	<ul style="list-style-type: none">▪ RGCN Encoder▪ DeDicom Decoder

Table of Contents

1. Abstract
 - 1.1 Motivation
 - 1.2 Results
 - 1.3 Availability and Implementation
2. Introduction
 - 2.1 Brief overview of GNNs working
3. Dataset
4. Related works
5. Model Overview/Methodology
 - 5.1 node2vec
 - 5.2 RGCN Encoder
 - 5.3 DistMult Decoder
 - 5.4 DeDicom Decoder
 - 5.5 MLP decoder
6. Experimental settings
7. Results and Observations
 - 7.1 Predictive Analysis
8. Future works
9. References

Abstract

Motivation: The treatment of complex diseases by taking multiple drugs becomes increasingly popular. However, Drug-Drug Interactions (DDIs) may give rise to the risk of unanticipated adverse effects and even unknown toxicity. Millions of people take upwards of five medications a day, but testing the side effects of such combinations is impractical. And lots of money and effort has to be put in to test whether a drug will cause these side-effects. The knowledge of Drug-Drug Interaction is often limited because these complex relationships are rare and are usually not observed in relatively small clinical testing. Therefore, it remains an important challenge to predict these kinds of relationships.

Results: Here we used a method that was built on the top of Z et.al. DECAGON work to predict drug-drug interactions by extracting the network structure features from DDI, DPI, PPI networks with graph neural networks (GNN) and a decoder that takes a pair of nodes and predict the type of edge between them. This approach constructs a pre-training with node2vec algorithm and then RGCN Encoder to map the structure as well as the meta edges present in our graph. We find that it automatically learns representations of drug-drug interactions, it achieves better performance than Z et al. DECAGON which had no pre-training.

Availability and implementation: Source code of our project has been provided with the report. And source code and pre-processed datasets provided by Z et al. are at: <https://github.com/mims-harvard/decagon>

Introduction

Drug-drug interactions (DDIs) can often occur when a drug is co-administered with another and multiple drugs, which will result in many adverse drug reactions (ADRs) that may cause injuries or deaths. Therefore, to alleviate DDIs the impact of unexpected pharmacological effects, it is critical to effectively identify potential DDIs, which can minimize unexpected ADRs and maximize synergistic benefits when treating a disease to some extent.

There are about 1,000 known side effects and 5,000 drugs on the market, making for nearly 125 billion possible side effects between all possible pairs of drugs. Most of these have never been prescribed together, let alone systematically studied.

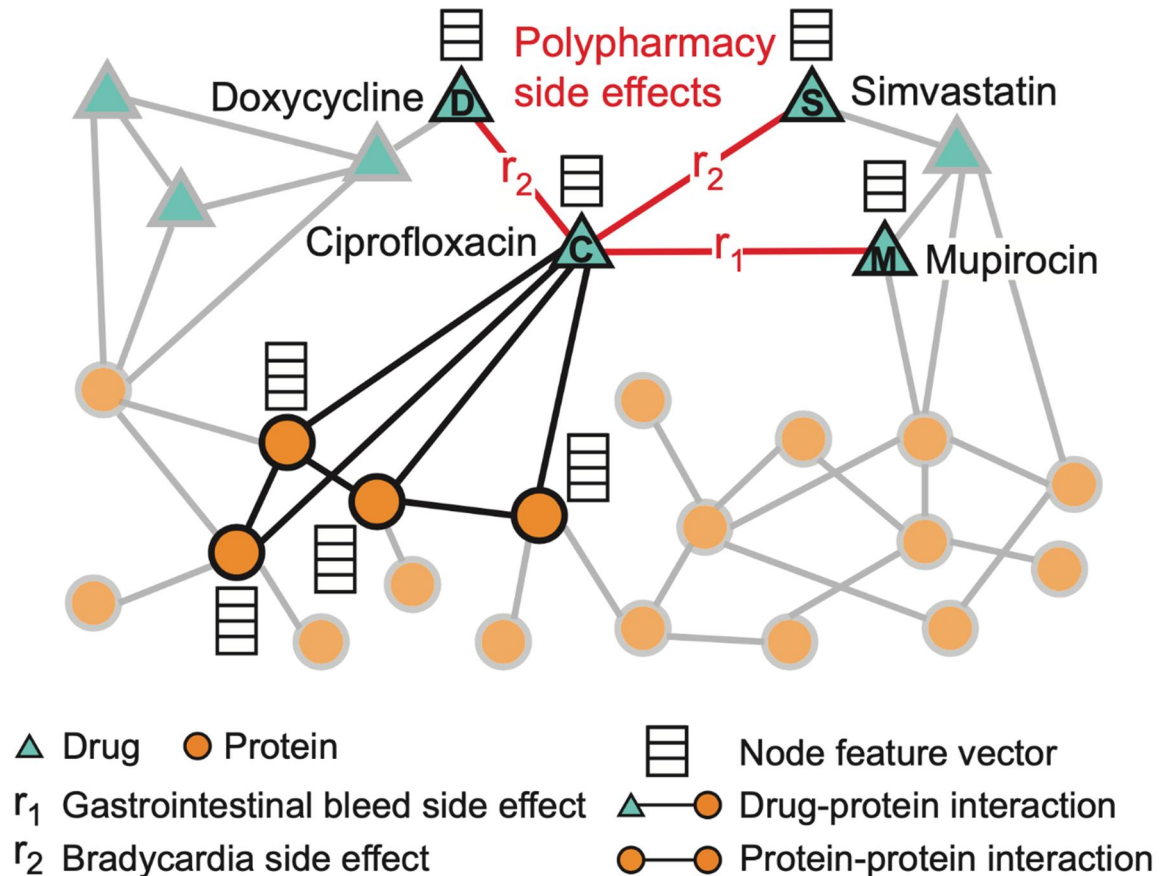


Figure 1: An example of polypharmacy side-effect credit Z et al.

We developed the DECAGON model for predicting side-effects of drug pairs but our approach initialised the embeddings with the help of node2vec. These embeddings are then given to the DECAGON model which capture the structure as well as the meta edge. Our model is inspired by RGCN [Michael et al.] model which are the class of GNNs specifically developed to deal with high multi relational data of realistic knowledge bases. We demonstrate the effectiveness of RGCN with node2vec pre-training combined with factorization model for link prediction such as DistMult, MLP Decoder, DeDicom Decoder.

Brief overview of the GNNs working:

GNNs work as they aggregate information from their neighbourhood for example, a node will get updated with the information present on itself and also the information from its neighbours. A new embedding of this node is formed which can be given to further downstream tasks such as node classification / link prediction and from that task a loss can be generated will be back-propped through the entire network. Through which we can learn how the neighbourhood aggregation happens in the first place.

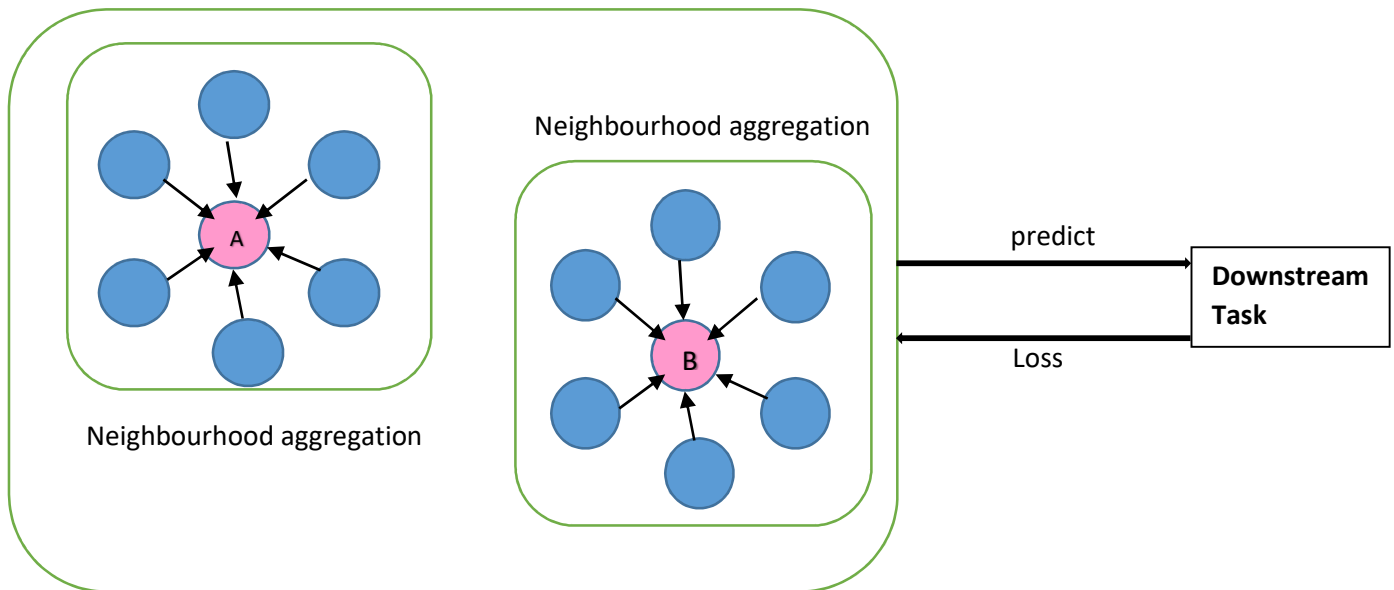


Figure 2: GNN working example

Dataset

We extracted pre-processed dataset provided by Z et al. Our resource includes all DrugBank interactions that met the following criteria:

1. The interactions between a drug and a single protein represent a target which is "protein group".
2. The interactions between a drug and a drug represent a side-effect category. There are a total of 1317 side-effects in our dataset.

In total, we extracted 19,726 interactions for 645 drugs and 19081 proteins. And there are about 10,767,298 edges in our network.

Related Works

Bi-Level Graph Neural Networks for biological link prediction:

The task of link prediction is common in the field of Medicines. Existence of a particular interaction between two biological molecules like Drugs, Proteins, etc. allow for novel medicines to be created to cure diseases.

Bi-Level GNNs are one such type of Neural Networks that can be used to perform link prediction tasks. The key idea is to fundamentally view the data as a bi-level graph, in which the highest level represents the interaction between biological entities (interaction graph), and each biological entity itself is further expanded to its intrinsic graph representation (representation graphs) in the lower level.

The two stages in Bi-Level GNNs:

1. **Lower Level Representation Graph Embedding:** In Stage-I, representation graphs' graph embeddings are generated. In general, this can be obtained through any graph embedding method. These can be embeddings that represent single level information about a given drug or hierarchical like a protein with amino acid level Graph information as Amino acids are the monomers that make up proteins.
2. **Higher Level Interaction Node Embedding:** Graph embeddings from Stage-I are used as initial node features for the interaction graph. This is motivated by the intuition that the lower level network presents a useful initial representation from which the higher level network further enhances for the given task. These representations can then be used for link prediction, link classification.

Our adaptation from Bi-GNNs for the task of Link Prediction:

We have an Encoder network which takes in Initial Node features for each node and uses Relational Graph Neural Networks to produce Embeddings. These represent the Low Level Features in Bi-GNN terminology. Then a Decoder Network composed of Multi-Layer Perceptron takes a pair of node embeddings obtained from Encoder Network, concatenates them and then tries to predict if a link between those nodes exist or not. These interactions can be drug-drug, protein-protein, drug-protein.

Decagon Model**Graph Convolutional Encoder**

- The initial node embeddings are first passed through an encoder, a **K-layer RGCN** [Relational Graph Convolutional Networks] network.
- As we are working with **different types of edges**, a simple GCN would not capture the embeddings effectively, so that is the reason why we use RGCN here.
- **RGCN computes** the resultant **node encoding**, by taking into consideration it's **neighbour's/sub-neighbour's embedding** based on the **edge type** they have.

Tensor Factorization Decoder

- The decoder model consists of a **global weight matrix R** , a **Diagonal Weight Matrix D_r** for every **Drug-Drug relation r** and a **Weight Matrix M_r** for all the **other relation type r** .
- After generating the node encodings through the RGCN encoder, they are passed to a decoder model along with the edge indexes and their types.
- The **score** of the connected nodes is calculated with **appropriate weight matrices** which are decided by the class of nodes.

If the **interacting nodes** are **both** part of **Drugs**, then the **weight matrix** is taken as **$D_r \times R \times D_r$** else if one or both of the nodes are part of **Protein** then the weight matrix is **M_r** .

Model Overview/Methodology

Node2vec

In order to create the initial embeddings for each node before passing it onto the Encoder Network, we use Node2Vec algorithm which learns low-dimensional representations for nodes in a graph by optimizing a neighbourhood preserving objective and uses random walks through a graph starting at a target node. It provides a way of balancing the exploration-exploitation trade-off which is obtained by adopting both DFS and BFS on the given graph.

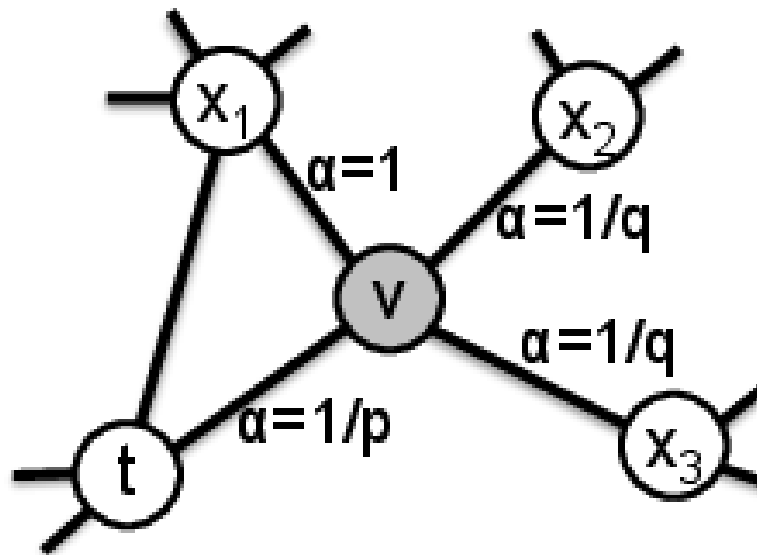
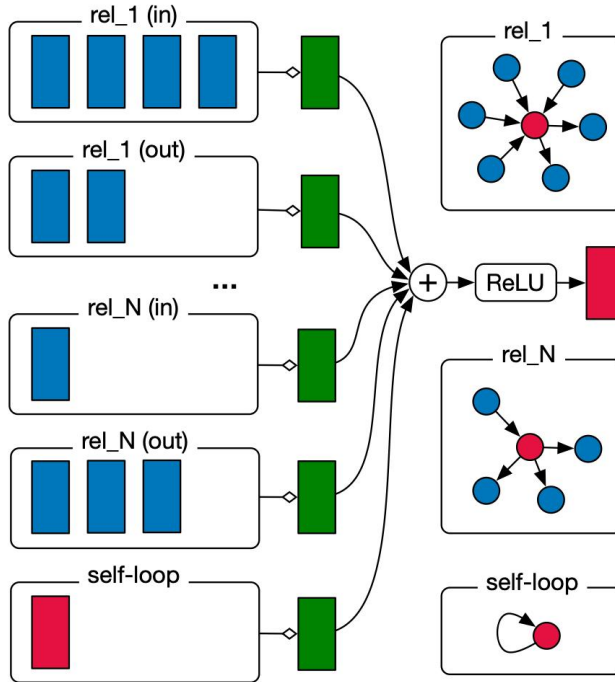


Figure 3: An illustration of random walk in node2vec

After transitioning to node v from t , the return hyper-parameter, p and the in-out hyper-parameter, q control the probability of a walk staying inward revisiting nodes (t), staying close to the preceding nodes (x_1), or moving outward farther away (x_2, x_3).

A high q would mean there is more exploration (DFS) while a high p would mean more of walking through nodes through BFS. The algorithm argues that the added flexibility in exploring neighbourhoods is the key to learning richer representations of nodes in graphs

RGCN Encoder



How RGCN differs from traditional graph convolutional networks is that it aggregates information from different nodes connected with different node types with different trainable parameter matrices. From this figure, we can see that there can be **N** relations in the graph, so we have **N** parameter matrices for each of those edge types.

Figure 4: An example of how a RGCN aggregates information from different types of edges.

We develop a non-linear, multi-layer convolutional graph neural network model RGCN that operates directly on graph G .

Let $R \in$ all the relations in our graph G .

In each layer, RGCN propagates latent node feature information across edges of the graph, while taking into account the type (relation) of an edge. A single layer of this neural network model takes the following form:

$$h_i^{(i+1)} = \sigma \left(\sum_{r \in R} \sum_{j \in N_i^r} \frac{1}{c_{i,r}} W_r^{(l)} h_j^{(l)} + W_0^{(l)} h_i^{(l)} \right)$$

where, N_i^r denotes the set of neighbour indices of node i under relation r .

$c_{i,r}$ is normalization constant.

$h_i^{(l)}$ is denotes the latent features of node(i) at l^{th} layer.

$W_r^{(l)}$ is the parameter matrix at layer l for relation R .

$W_0^{(l)}$ is the parameter matrix at layer l for how much information a node wants from itself.

To update the features of node(i), we take the matrix multiplication with the current features of node(i) and another matrix multiplication with the neighbours of node(i) and aggregates those neighbourhood features by taking the mean over them. We do for each relation in R .

DistMult Decoder

DistMult is a factorisation decoder that takes in embeddings of two nodes and gives a probability that a relation r exists between those nodes.

$$DistMult(A, r, B) = \sigma(z_A^T W_r z_B)$$

DeDicom Decoder

DeDicom Decoder is also a tensor factorisation decoder which is specially designed to work when a relation r has comparatively less ratio than other relations.

$$DeDicom(A, r, B) = \begin{cases} z_A^T D_r R D_r z_B & \text{if } z_A \text{ and } z_B \text{ are drugs} \\ z_A^T M_r z_B & \text{any other case} \end{cases}$$

here D_r is the diagonal matrix for each relation r

R is a common matrix of size $d \times d$.

M_r is a parameter matrix of $d \times d$.

Some of the drug relations happens very rarely so our intuition is that their parameter matrices will get updated comparatively less number of times than for other relations which may be huge in number. How DeDicom helps us to solve this issue is that we can a common matrix R that gets updated for any side-effect category.

MLP Decoder

For MLP decoder, we have a Multi-Layer Perceptron (MLP) that takes in two embeddings z_A and z_B , concatenates them and passes them through the network which then produces R class scores each representing the probability of a relation r existing between A and B .

Experimental Settings

For the encoder, we have kept the number of layers as 2 that means in an overall framework a node is collecting information from its 2-hop away neighbours.

We find that if we go deeper in the number of layers that mean a node will be getting updated from a huge chunk of our graph. It would also mean the nodes around it are also collecting information from the same neighbourhood. That can lead to over-smoothing problems (well known in GNNs) which implies most of the nodes in our graph have similar information and it is going to be difficult for the decoder to distinguish between them.

We also experimented with different types of initialisations which include random initialisation, one-hot initialisation and node2vec initialisation.

We also find that the hidden layer representations for nodes does not need to be more than 64 as it is giving us SOTA results.

Hyper-Parameter	Values
Initial node feature size	10 (random)/ 576 (one-hot)/ 128 (node2vec)
Batch size	128
Walk_Length	16
No. of steps	32
Data loader	GraphSAINTRandomWalkSampler
Learning rate	1e-3
Optimiser	SGD
Momentum	0.9
Weight Decay	5e-4
LR Scheduler	None
Epochs	120 (random)/ 50 (one-hot)/ 60 (node2vec)
Loss Function	Cross Entropy

Table 1: Hyper-parameters.

Results and Observations

From table 2, we can see the performance of different models and also see that our **node2vec model is out performing** the **Decagon** model because of better initialisation. **Bi-GNN** is out performing all of the models that is because it also takes into accounts of structural composition of different proteins and chemical composition of different drugs i.e., it has better initialisations because they are based on **structural evidence** of the nodes themselves.

Model	Initialisation	AUROC
Decagon	Random	0.872
Bi-GNN	Protein structure + Drug Structure	0.933
Ours*	Random	0.680
	One-hot	0.542
	Node2vec	0.904

Table 2: AUROC comparison for different models.

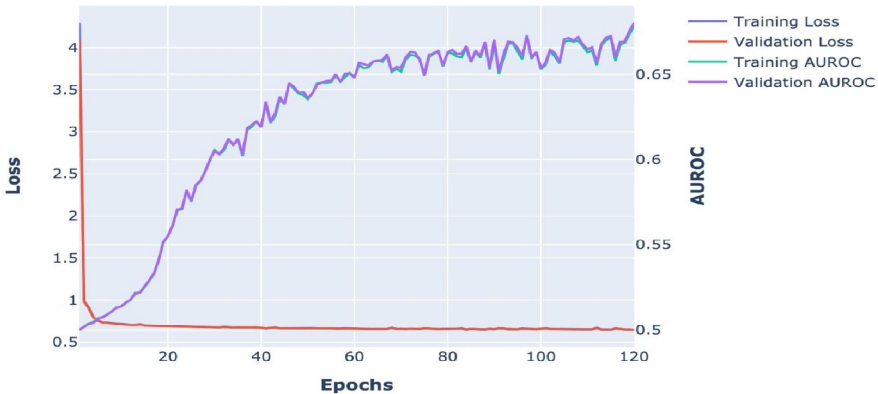
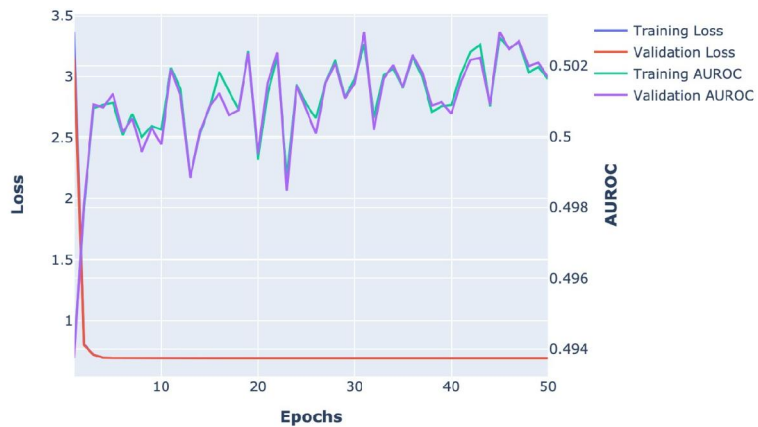
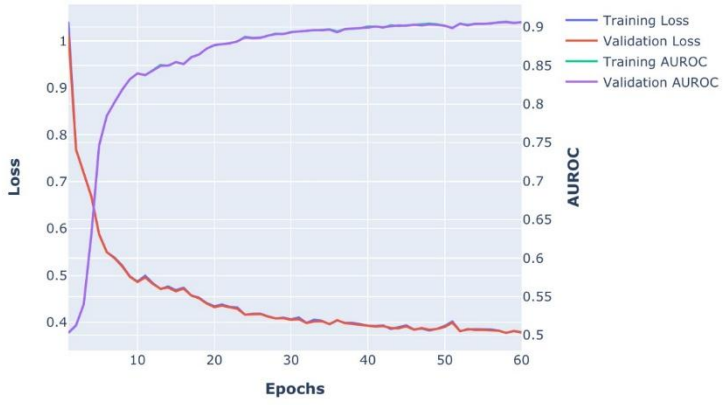
Random	<div>Loss/AUROC plot</div>  <p>Loss/AUROC plot for Random baseline. The plot shows Training Loss (blue line) and Validation Loss (red line) on the left y-axis (0.5 to 4.0), and Training AUROC (green line) and Validation AUROC (purple line) on the right y-axis (0.5 to 0.65). The x-axis represents Epochs (0 to 120). Training Loss decreases from ~4.0 to ~0.7. Validation Loss decreases from ~4.0 to ~0.7. Training AUROC increases from ~0.5 to ~0.65. Validation AUROC increases from ~0.5 to ~0.65.</p>
One-hot	<div>Loss/AUROC plot</div>  <p>Loss/AUROC plot for One-hot baseline. The plot shows Training Loss (blue line) and Validation Loss (red line) on the left y-axis (1 to 3.5), and Training AUROC (green line) and Validation AUROC (purple line) on the right y-axis (0.494 to 0.502). The x-axis represents Epochs (0 to 50). Training Loss decreases from ~3.5 to ~0.5. Validation Loss decreases from ~3.5 to ~0.5. Training AUROC increases from ~0.5 to ~0.502. Validation AUROC increases from ~0.5 to ~0.502.</p>
Node2vec	<div>Loss/AUROC plot</div>  <p>Loss/AUROC plot for Node2vec baseline. The plot shows Training Loss (blue line) and Validation Loss (red line) on the left y-axis (0.4 to 1.0), and Training AUROC (green line) and Validation AUROC (purple line) on the right y-axis (0.5 to 0.9). The x-axis represents Epochs (0 to 60). Training Loss decreases from ~1.0 to ~0.4. Validation Loss decreases from ~1.0 to ~0.4. Training AUROC increases from ~0.5 to ~0.9. Validation AUROC increases from ~0.5 to ~0.9.</p>

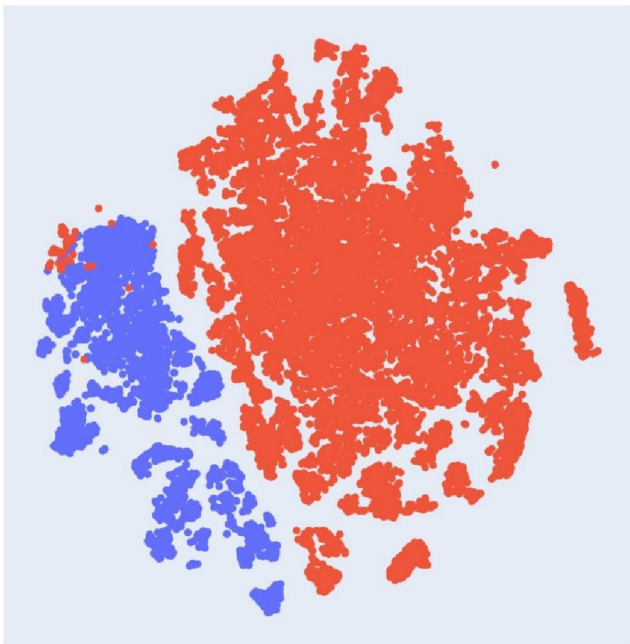
Table 3: Loss/AUROC curves of train and validation graphs.

From table 3, we can visualise how the models trained over epochs.

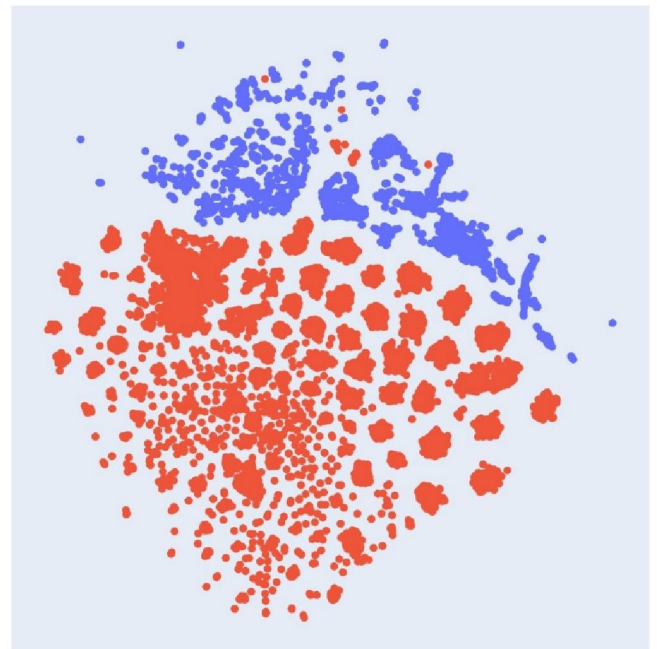
1. For the random initialisation, the loss at the start is really high which was expected because they are just randomly initialised features. As the training progresses it stabilises at around 0.68 AUROC.
2. For one-hot initialisation, we believe that our model is over-fitting to node initialisation because some of the one-hot vectors for many nodes maybe similar.
3. For node2vec initialisation, the training loss at the start was low compared to others because of the quality embeddings node2vec provides us, and also they achieve much higher performance than other initialisation techniques.

Predictive Analysis

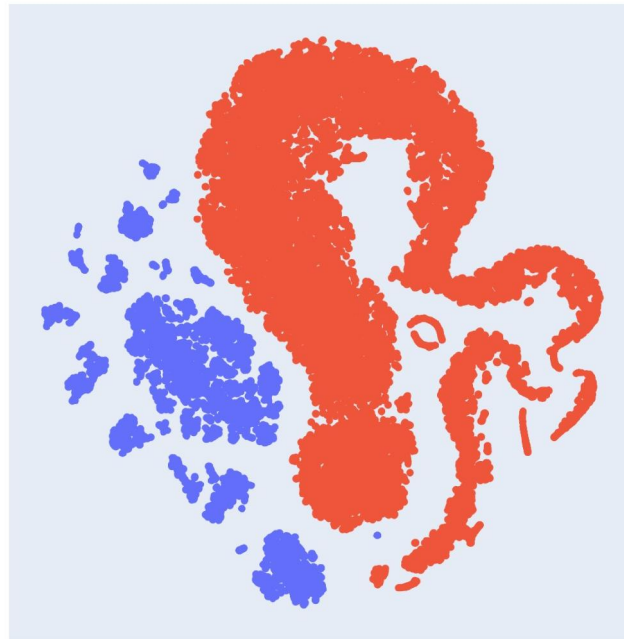
For predictive analysis, our task was to visualise the embeddings for each node that we have in our graph. For this employed **t-SNE** dimensionality reduction algorithm rather than **PCA** because **t-SNE** focuses more on local relationships and reduces those relationships correctly even in a smaller dimension, whereas **PCA** focuses more on global relationships and tries to encode all the information even in lower dimension.



t-SNE for random initialisation



t-SNE for one-hot initialisation



t-SNE for node2vec initialisation

Figure 5: t-SNE embedding of final node features of our graph

1. From the above graphs, we see that the embeddings for **random initialisation** are well separated for drug and protein nodes, but if we try to visualise the number of drug clusters within, it seems that it has grouped a lot of clusters and is not able to distinguish between them, that explains the low performance.

And also if we try to visualise the protein clusters, it seems that it has formed one single cluster for all the protein nodes.

2. For the t-SNE graph of **one-hot initialised** model, we see that it has formed lots of individual drug and protein clusters and seems to have over fit to our data.
3. For the t-SNE graph of **node2vec initialised** model, we see that it has not only learned to separate drug and protein nodes but has also learned to separate the clusters of different drugs and clusters of different proteins in the sub space.

Conclusion

We proposed an effective and robust method to predict the potential DDIs by utilizing the DDI network information without considering the drug properties (i.e., drug chemical and biological properties). The method should also be useful in other DDI-related scenarios, such as the detection of unexpected side effects, and the guidance of drug combination.

Future Works

- ❖ Different GNN Models like Heterogeneous Graph Transformers may lead to better results.
- ❖ Reinforcement learning: Allow agents to use node embedding information to make decisions.
- ❖ Using pre-trained node embeddings from a different model on different tasks may lead to better results.
- ❖ Extract features for the structure of nodes (proteins, drugs) and use those features as a starting point in our model

References

1. Wishart DS, Knox C, Guo AC, Shrivastava S, Hassanali M, Stothard P, Chang Z, Woolsey J. Drugbank
2. Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, Max Welling (2017) Modeling Relational Data with Graph Convolutional Networks NIPS
3. Bordes, A.; Usunier, N.; Garcia-Duran, A.; Weston, J. and Yakhnenko, O. 2013. Translating embeddings for modeling multi-relational data. In NIPS.
4. Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. In ICLR'18.
5. William L. Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In NeurIPS'17
6. Marinka Zitnik, Monica Agrawal and Jure Leskovec. 2018. Modeling polypharmacy side effects with graph convolutional networks OXFORD