SAPIENZA
UNIVERSITÀ DI ROMA

Graph Neural Networks: Protein-Protein Link Prediction

Data Science Honor Program 2021-2022

Nicola Calabrese 1797714

1 Introduction

Graph Neural Networks (GNNs) are a specific class of neural networks which work very well when the data can be represented and processed as graphs. For this reason they are naturally suitable for modelling molecular systems. Some of these applications are: molecular property prediction, molecular generation, molecular dynamics simulation and docking [1]. Research didn't stop here and, due to the fact GNNs can be used for link prediction, i.e. predicting the existence of an edge between two arbitrary nodes in a graph, researchers tried to use them for predicting the interactions between proteins. This application can be fundamental for understanding biological processes and maybe for revealing disease mechanisms and consequently develop most effective treatments.
The difficulty rises from the fact that link prediction methods are usually based on some network similarity, instead interacting proteins are not necessary similar and similar proteins do not necessarily interact .
In this project I explored different architectures of GNNs with the aim of understand which ones are more able to learn more hidden features and relations between proteins, reaching higher accuracy.

2 Dataset

The data used come from the Biological General Repository for Interaction Datasets (BioGRID), which is a biological database of protein-protein interactions, genetic interactions and chemical interactions for different organism species. In particular I used *BIOGRID-ORGANISM* which contains all well known interactions between proteins in the human body.

The dataset is formed in this way:

- number of nodes: 19776
- number of edges: 936634
- largest component: 19776

The number of nodes in the largest component is equal to the total number of nodes in the graph, this means that there aren't isolated nodes.
For the number of edges some pre-processing steps are needed. After removing self-loops and multi-edges the number of edges becomes 699259.

3 Implementation

3.1 Node2Vec

Due to the fact the dataset provides only edges between nodes, and the nodes have no features, it is useful to have a numerical representation of the network structure, Node2Vec [2] suits perfectly for this case.
Node2Vec is an algorithm that allows the user to map nodes in a graph G to an embedding space. The idea is to preserve the original structure of the graph, meaning that nodes which are similar within the graph will have similar embeddings in the embedding space.
The algorithm is based on the concept of *random walk*, which is a type of stochastic process, which explores the relationship to each step that you would take and its distance from the initial starting point. To introduce a bias to influence the random walks, two parameters are used:

- p: indicates the probability of a random walk getting back to the previous node.

- q: indicates the probability that a random walk can pass through a previously unseen part of the graph.

It can be visually seen in the following figures.

$$\alpha_{pq}(t,x) = \begin{cases} \frac{1}{p} & \text{if } d_{tx} = 0 \\ 1 & \text{if } d_{tx} = 1 \\ \frac{1}{q} & \text{if } d_{tx} = 2 \end{cases}$$

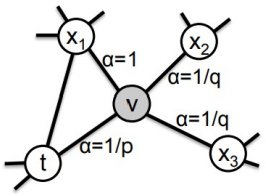

Figure 1: Random walk procedure in Node2Vec.

3.2 Graph Architectures

3.2.1 Graph Auto-Encoders (GAEs)

Graph Auto-Encoders applies the idea of Autoencoders on graph-structured data [3]. First an Encoder is used to map the graph into a latent space Z and then use a Decoder to reconstruct the adjacency matrix.

The GAE model has a single-GCN layer as Encoder, which generates a latent variable Z directly, and an inner product as Decoder. The steps are the following:

$$\hat{A} = \sigma(ZZ^T), \ with \ Z = GCN(X, A)$$

Deep GAEs Deep GAEs are just an "*enhanced*" version of GAEs. The idea is simple: instead of using only one GCN layer to generate the latent variable z, more GCN layers are stacked one after each other. Sequence of convolution operations can lead to more informative latent representation. Therefore, if the embedding captures more information from the input, the output will have a better performance.

Linear Decoder Instead of using just a simple inner product as Decoder, it is possible to have a more complex Decoder made of linear transformation $y = xA^T + b$ of the data coming from the latent space. Inner product could calculate the cosine similarity of two vectors, which is useful when we want a distance measure that is invariant to the magnitude of the vectors, it's easy to compute and easy to interpret, but sometimes the embedding has a lower dimensionality than the input, so it cannot contain all the information, thus, having linear layers as Decoder can compensate this loss of information and lead to better data reconstruction.

3.2.2 Variational Graph Auto-Encoders (VGAEs)

Variational Graph Auto-Encoders are the probabilistic version of GAEs, the idea is to embed the input X to a distribution rather than a point. Then a random sample Z is taken from the distribution rather than the one generated from encoder directly [4].
Learning the distribution of the latent space means that it is possible to generate new data from the source dataset. Traditional GAEs instead, could only generate graphs that are similar to the original inputs.

The Encoder of a VAE takes a data point x and produces a distribution usually indicated as $q_\phi(z|x)$, which is usually parametrized as a Multivariate Gaussian. Therefore, the Encoder generates μ and $log\ \sigma^2$ as outputs, the mean and standard deviation of the Gaussian distribution. The lower-dimensional embedding z is sampled from this distribution as $z = \mu + \sigma * \epsilon$, where $\epsilon \sim\ N(0, 1)$. The Decoder instead takes an embedding z and produces the output $\hat{X}$, from $p_\theta(x|z)$.

3.2.3 Adversarially Regularized Variational Graph Autoencoder (ARGVA)

Embedding algorithms typically focus on preserving the topological structure or minimizing the reconstruction errors of graph data, but they have mostly ignored the data distribution of the latent space from the graphs.
ARGVA instead is a adversarial framework which enforces the latent space to match a prior distribution [5].
To learn a robust graph representation an adversarial training scheme is incorporated. Its aim is to discriminate if the latent embeddings are from a real prior distribution or from the graph encoder. The figure below shows the structure of a ARGA (adversarially regularized graph autoencoder):

- In the upper part we have the usual graph autoencoder which tries to reconstruct the graph structure A given the embedding z,
- In the bottom part an adversarial network is trained to discriminate if a sample is generated from the encoder or from a prior distribution.

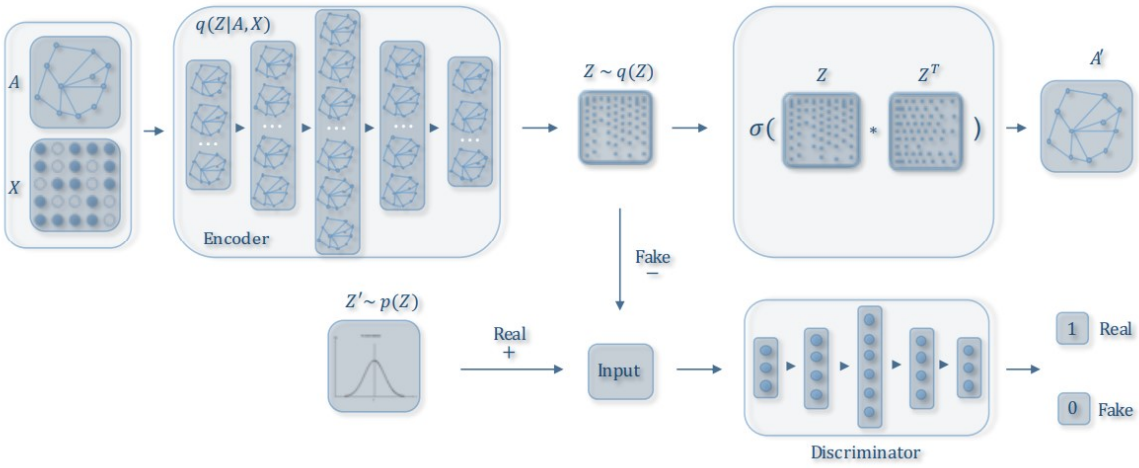

Figure 2: Adversarially regularized graph autoencoder (ARGA)

The variational version is similar: as described above, instead of having $z \sim\ q(z)$, the embedding z is generated from μ and σ as $z = \mu + \sigma * \epsilon$, where $\epsilon \sim\ N(0, 1)$.

4 Results

Performance of each graph architecture are measured by AUC (Area Under The Curve), which tells how much the model is capable of distinguishing between classes. Higher the AUC, the better the model is at predicting 0 classes as 0 and 1 classes as 1. Here the classes are: "presence of link" and "absence of link" (negative link).

The figure below shows the AUC for 4 different configurations of GAEs, described previously in the implementation section:

- GAE
- GAE with linear decoder
- DeepGAE
- DeepGAE + linear decoder

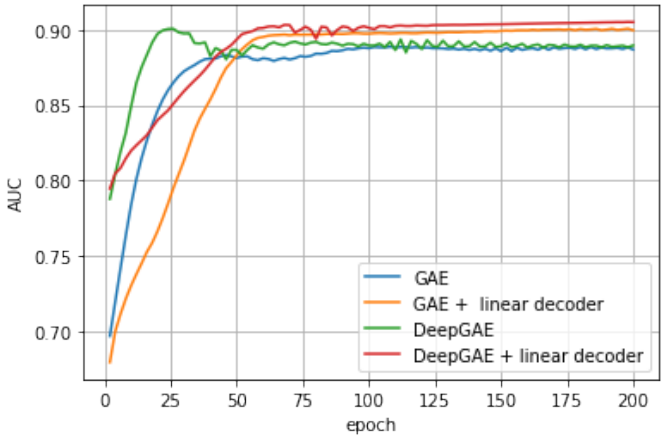

Figure 3: AUC comparison for different GAE variations

It seems there is no huge difference in between GAE and DeepGAE, so using a single Graph Convolutional Layer or more, but when the decoder is not only an inner product but linear layers are used AUC improves.

The best configuration, DeepGAE + linear decoder, is now compared with more complex architectures presented before:

- VGAE
- ARGVA

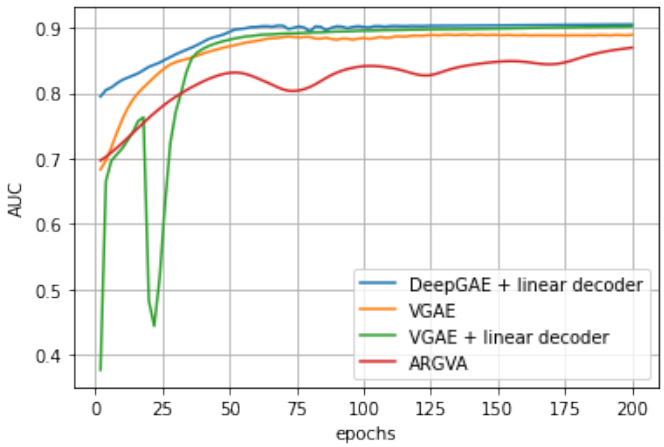

Figure 4: AUC comparison for GAE, VGAE and ARGVA

VGAE + linear decoder reaches the same performances even if it needs more epochs to reach the stability, in the first epochs it oscillates quite a lot, meaning it has some difficulty in learning the proper distribution of the latent space. 200 epochs instead were not sufficient for ARGVA to reach the stability, it seems the performance can still increase but from a visual inspection the rate of increasing seems not high enough to reach the 0.91 as DeepGAE and VGAE both with linear decoder.

5 Conclusions

This project presented an interesting application of GNNs: protein-protein links prediction. Different architectures were introduced and tested: from simple GAEs , passing through variational GAEs, ending with adversarial training like ARGVA. Unfortunately the original graph is quite huge and deeper architectures would require a lot of time to train without a dedicated GPU, furthermore no features were attached to the nodes. But even with small architectures, extracting embeddings with topological algorithm like Node2Vec, and few minutes of training on CPU, a good value of AUC was reached: 0.91 for the best model. This is going to be a solid benchmark result for the future works.

References

[1] Yuyang Wang, Zijie Li, and Amir Barati Farimani. *Graph Neural Networks for Molecules*. 2022. DOI: 10.48550/ARXIV.2209.05582. URL: https://arxiv.org/abs/2209.05582.

[2] Aditya Grover and Jure Leskovec. *node2vec: Scalable Feature Learning for Networks*. 2016. DOI: 10.48550/ARXIV.1607.00653. URL: https://arxiv.org/abs/1607.00653.

[3] Fanghao Han. *Tutorial on Variational Graph Auto-Encoders*. 2019. URL: https://towardsdatascience.com/tutorial-on-variational-graph-auto-encoders-da9333281129.

[4] Thomas N. Kipf and Max Welling. *Variational Graph Auto-Encoders*. 2016. DOI: 10.48550/ARXIV.1611.07308. URL: https://arxiv.org/abs/1611.07308.

[5] Shirui Pan et al. *Adversarially Regularized Graph Autoencoder for Graph Embedding*. 2018. DOI: 10.48550/ARXIV.1802.04407. URL: https://arxiv.org/abs/1802.04407.