Predicting of Protein Interactions using GCNs

Abderrazzak El Khayari
Zidanelkhayari@gmail.com

Rihab Ziani (Phase1)
rihabzn@gmail.com

Deepak Rastogi
rastog01@ads.uni-passau.de

Vishvapalsinhji Ramsinh Parmar
parmar03@ads.uni-passau.de

1 PROBLEM STATEMENT

1.1 Introduction to GCNs

- **What is a Graph Convolutional Network**

Nowadays, models on graph neural networks have emerged in machine learning and other related areas, and demonstrated the superior performance in various problems.

Graphs naturally arise and appear in many real-world applications, such as social analysis, fraud detection, predicting protein protein interactions and traffic prediction.

In machine learning, GCNs are considered a very powerful neural network architecture on graphs. In fact, they are so powerful that even a randomly initiated 2-layer GCN can produce useful feature representations of nodes in networks.

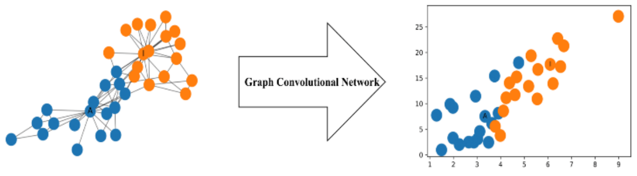

Figure 1: Illustration of a 2-dimensional representation of each node in a network produced by GCN.

The relative nearness of nodes in the network is preserved in the 2-dimensional representation even without any training [1].

A graph convolutional network (GCN) is a neural network that operates on graphs. Given a graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$, a function of signals/features takes as input:

- A feature description $\boldsymbol{x_i}$ for every node $\boldsymbol{i}$; summarized in a $\boldsymbol{N} \times \boldsymbol{D}$ feature matrix $\boldsymbol{X}$ ($\boldsymbol{N}$: number of nodes, $\boldsymbol{D}$: number of input features)
- A representative description of the graph structure in matrix form; usually in the form of an adjacency matrix $\boldsymbol{A}$

and produces a node-level output $\boldsymbol{Z}$ (an $\boldsymbol{N} \times \boldsymbol{F}$ feature matrix, where $\boldsymbol{F}$ is the number of output features per node).

A hidden layer in the GCN can thus be written as a non-linear function:

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

with $\boldsymbol{H}^{(0)} = \boldsymbol{X}$ and $\boldsymbol{H}^{(L)} = \boldsymbol{Z}$ (or $\boldsymbol{z}$ for graph-level outputs), $\boldsymbol{L}$ being the number of layers. The specific models then differ only in how the propagation $\boldsymbol{f}(\cdot,\cdot)$ is chosen and parameterized.

- **Propagation Rule**

One of the simplest possible propagation rule is:

$$f(H^{(l)}, A) = \sigma(AH^{(l)}W^{(l)})$$

where $\boldsymbol{W}^{(l)}$ is a weight matrix for the l-th neural network layer and $\boldsymbol{\sigma}(\cdot)$ is a non-linear activation function.

This simple model has two significant shortcomings: multiplication with $\boldsymbol{A}$ means that, for every node, we sum up all the feature vectors of all neighboring nodes but not the node itself (unless there are self-loops in the graph). As a solution, we add the identity matrix to $\boldsymbol{A}$ to enforce self-loops in the graph.

The other major limitation is that nodes with large degrees will have large values in their feature representation while nodes with small degrees will have small values, which can lead to issues with exploding gradients and make it harder to train using algorithms which are sensitive to feature scaling. $\boldsymbol{A}$ is typically not normalized and therefore the multiplication with $\boldsymbol{A}$ will completely change the scale of the feature vectors. To get rid of the problem, we normalize $\boldsymbol{A}$ such that all rows sum to one, i.e. $\boldsymbol{D}^{-1}\boldsymbol{A}$, where $\boldsymbol{D}$ is the diagonal node degree matrix. Multiplying with $\boldsymbol{D}^{-1}\boldsymbol{A}$ now corresponds to taking the average of neighboring node features. In practice, dynamics get more interesting when we use a symmetric normalization, i.e. $\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}$ (as this no longer amounts to mere averaging of neighboring nodes). Combining these two, we essentially arrive at the propagation rule:

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

with $\hat{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$, where $\boldsymbol{I}$ is the identity matrix and $\hat{\boldsymbol{D}}$ is the diagonal node degree matrix of $\hat{\boldsymbol{A}}$ [2].

1.2 A short review on state-of-the-art for PPIs

Proteins are chains of amino acid residues that fold into a three-dimensional structure that gives them their biochemical function. Proteins perform their function through a complex network of interactions with other proteins. Prediction of the interaction between two proteins is an important research problem with applications in genetic diseases and pharmacological research that allow us to understand the behavioral processes of life, preventing diseases, and developing new drugs.

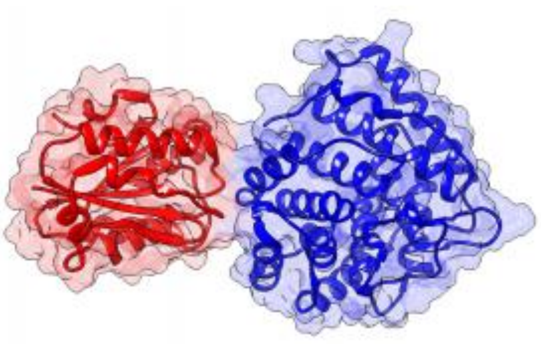

Figure 2: Two proteins in their bound formation [3].

There are many computational methods that have been proposed as complementary to experimental methods to predict protein-protein interactions, one is using Graph Convolutional Networks (GCNs) which will be tackled in our project.

1.3 Formal definition of the problem

Given two proteins, we predict in our work the probability that they physically interact in a cell, using an operator that generates an embedding for any pair of nodes.

In link prediction, we have a community of nodes with a certain fraction of edges removed, and we would like to predict these missing edges.

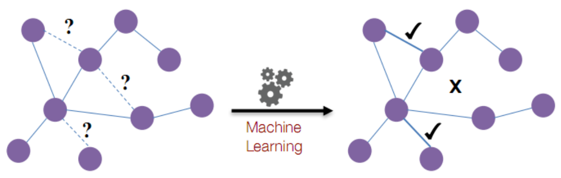

Figure 3: Illustration of prediction protein protein interaction using machine learning.

In this demo, we take the yeast protein-protein interaction network (download the preprocessed network) and use the network to build a model for predicting new protein-protein interactions. We formulate this prediction task as a link prediction problem on unweighted and undirected networks and use a graph convolutional neural network to solve the task.

- **How to predict missing edges**

We need to start with some basic assumptions about what elements in our data might predict whether two proteins will interact at a later date. We'll begin with the assumptions that the following elements increase the probability that two protein interacts:

- Given u and v, we define an operator g that generates an embedding for pair (u, v) :
  $$z_{(u,v)} = g(u, v)$$
  Such that $g(u, v) = z_u * z_v$
- By hiding 20% of the network edges, our model learns the embedding using the rest 80% edges and predict the most likely edges which are not observed in the training dataset.
  We use the activation function[1] in the output layer: $f = \frac{1}{1+e^{-\beta Z}}$
  such that:
  β is the final learning weights.
  $Z = <1, Z_1, Z_2, .. Z_n>$,
  $Z_i \in \{\#Common\ neighbors, \#\ Jaccard's\ coefficient\}$
  Jaccard Similarity[2]: $J = \frac{N_X \cap N_Y}{N_X \cup N_Y}$, where N_X and N_Y are the interaction partners of X and Y.

1 http://www.cscjournals.org/manuscript/Journals/IJAE/Volume1/Issue4/IJAE-26.pdf

2 https://deepai.org/machine-learning-glossary-and-terms/jaccard-index

REFERENCES

[1] Tobias Skovgaard Jepsen, *How to do Deep Learning on Graphs with Graph Convolutional Networks,* Available from: https://towardsdatascience.com/how-to-do-deep-learning-on-graphs-with-graph-convolutional-networks-7d2250723780

[2] THOMAS KIPF, *GRAPH CONVOLUTIONAL NETWORKS,* Available from: https://tkipf.github.io/graph-convolutional-networks/

[3] Alex Fout, Jonathon Byrd, Basir Shariat and Asa Ben-Hur. *Protein Interface Prediction using Graph Convolutional Networks*. Department of Computer Science, Colorado State University 2017.