MULTI-POOL CONVOLUTION FOR GRAPH REPRESENTATION LEARNING

A Project Report

submitted by

AASHAY BHUPENDRA DOSHI

in partial fulfilment of the requirements
for the award of the degree of

MASTER OF TECHNOLOGY

DEPARTMENT OF CHEMICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

MAY 2020

CERTIFICATE

This is to certify that the thesis titled **MULTI-POOL CONVOLUTION FOR GRAPH REPRESENTATION LEARNING**, submitted by **AASHAY BHUPENDRA DOSHI**, to the Indian Institute of Technology, Madras, for the award of the degree of **Bachelors and Masters of Technology**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

(Dr. Pratyush Kumar)
Research Guide
Dept. of Computer Science and Engineering
Indian Institute of Technology Madras,
Chennai - 600 036

(Dr. Sridharakumar Narasimhan)
Research Co-Guide
Dept. of Chemical Engineering
Indian Institute of Technology Madras,
Chennai - 600 036

Place: Chennai

Date: 15th June 2020

ACKNOWLEDGEMENTS

I would like to thank both Dr. Pratyush Kumar and Dr. Sridharakumar Narasimhan for agreeing to guide me and providing me with the freedom to explore my research interests.

ABSTRACT

KEYWORDS: Semi-Supervised Learning, Representation Learning, Relational Learning, Graph Classification, Node Classification, Deep Learning, Graph Convolutional Networks

In many real-life applications, entities in an environment are not independent but rather influenced by each other through their interactions. Such relational datasets are popularly modeled as graphs where the entities make up the node, and the edges represent an interaction. Learning algorithms for graph classification and node classification on such relational data is different from conventional machine learning algorithms which ignore the relational in-formation and assume samples (entities) are drawn from an IID. Relational learning algorithms for node and graph classification should adhere to the structure of the data and capture complex correlations between the node and its neighbors for the classification task. Given a graph where every node has specific attributes associated with it, and some nodes have labels associated with them. Collective Classification is the task of assigning labels to every unlabeled node using information from the node as well as its neighbors. Graph Classification is the task of identifying the correct labels of the graph. From a broader perspective, graph classification is the task of identifying the correct representation for the entire graph. It is often the case that a node is not only influenced by its immediate neighbors but also by it's higher order neighbors, multiple hops away. Variants of Graph Convolutional Neural Networks (GCNs) are the state-of-the-art neural network architectures for these classification problems. GCNs are end-to-end differentiable variations of recursively defined graph kernels that aggregate and filter multi-hop neighborhood information. GCNs are essentially operations developed to perform convolutional operation on non-euclidean data. In addition to convolution, similar to convolutional neural networks, graph classification tasks also require pooling operation to generate a single representation of the entire graph, further bridging the gap between images (Convolution Models) and graphs. There are a large number of pooling algorithms which use a plethora of methods from node clustering,

edge pooling all the way to using a score function with a threshold. This combination of convolution and pooling enables us to port almost all the convolutional neural network models to graph classification and regression problems; and a number of models have been developed using the same principal. Despite these advancements, accuracy and interpretability, along with node information morphing in deep networks, are still issues that plague any graph model. Another issue with node and graph embeddings is large networks inherently require deeper networks so that the model has a larger receptive field which is closer to the true network size. Garnering more from the world of image and text processing, we here represent Multi-pooling convolutions which takes this concept of pooling and uses it as an attention mechanism to generate sub-graphs for convolution. These sub-graphs can also be seen as hidden structures within the graph (e.g.: functional groups in an organic compound) and can be used for interpreting how the network classifies nodes and graphs. This is analogous to attention maps in natural language processing or class activation maps and receptive fields in convolutional neural networks. We extensively evaluate a the proposed models on datasets from biology, bibliography, sociology and finance domains under transductive and inductive learning setup.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

SSL	Semi-Supervised Learning
GRL	Graph Representation Learning
CC	Collective Classification
PSD	Positive-Semi Difinite
IID	Independent and Identicaly Distributed
PCA	Principal Component Analysis
ReLU	Rectifier Linear Unit
LSTM	Long Short Term Memory
CNN	Convolutional Neural Network
GNN	Graph Neural Network
GCN	Graph Convolution Network/Operator
GraphSAGE	Graph Sample and Aggregate
GAT	Graph Attention Network/Operator
GIN	Graph Isomorphism Network
SAGE Pool	Self-Attention Graph Pooling
Diff Pool	Differential Graph Pooling
MPNN	Message Passing Neural Network
WL	Weisfeiler-Lehman Kernel
GPU	Graphic Processing Unit

CHAPTER 1

Introduction

Many real-world problems are represented by using graphs. For example, given a graph of a chemical compound, we want to determine whether it causes a gene mutation or not. As another example, given a graph of a social network, we want to predict a potential friendship that does not exist but it is likely to appear soon. Many of these questions can be answered by using machine learning methods if we have vector representations of the inputs, which are either graphs or vertices, depending on the problem. In machine learning, feature learning or representation learning is a set of techniques that allows a system to automatically discover the representations needed for feature detection or classification from raw data. An efficient representation of data is critical for analysis and learning tasks. Efficient in terms of space allows data processing algorithms to scale and efficient in terms of in-formation allows processing algorithms to effectively uncover the different explanatory factors behind the variation in data. Devising learning algorithms explicitly to derive efficient representations for the end applications is crucial as it allows to conveniently encode general priors in the data like smoothness, sparsity, clustering, manifolds, mul-tiple explanatory factors, etc. Manifold learning and the more generic Representation learning are the two sub-fields of machine learning that focus on finding efficient data representations for ma-chine learning tasks with an intent to uncover the structure in the data. Manifold learn-ing methods are dimensionality reduction techniques that find a dense new coordinate system in low-dimensional regions of the original data, whereas representation learning methods can also produce sparse over-complete features in a high dimensional space with more dimensions than the input. Matrix (Tensor) factorization and Neural networks are two powerful and commonly used representation learning methods. In the current era of the burgeoning field of Deep Learning, deep neural net-work architectures have become the unanimous choice of representation learning in a majority of the domains for large data.

1.1 Network Representation Learning

In many real-life applications such as social networks, citation networks, protein interaction networks, etc., the entities in an environment are not independent but rather influenced by each other through their interactions. Network representation learning (NRL),focuses on extracting useful features in such complex networked data. Network data,unlike conventional data, does not follow the assumption that samples are drawn from an independent and identically distributed (IID) and additionally have some inherent structure. Network data may contain homogeneous or heterogeneous nodes which may exhibit one or more (un)directed homophilous or heterophilous relationships between nodes or even both. Network data can also contain one or more contextual features associated with nodes or relations. Representation learning algorithms for networks should preserve the structure in the data explicitly and encode the necessary priors required for the end task. Depending on the task, representations are learned for nodes, relations and the entire graph as such.Such network data or relational data have been popularly modeled as graphs where the entities make up the node, and the edges represent an interaction. The use of graph-based learning algorithms has increasingly gained traction owing to their ability to model structured data. Categorizing such entities requires extracting relational information from their multi-hop neighborhoods and combining that efficiently with their features. Summarizing information from multiple hops is useful in many applications where there exists semantics in local and group level interactions among entities. Thus,defining and finding the significance of neighborhood information over multiple hops becomes an essential aspect of the problem.

1.2 Collective Classification and Graph Classification

Given a graph where every node has specific attributes associated with it, and some nodes have labels associated with them, Collective Classification (CC) is the task of assigning labels to the remaining unlabeled nodes in the graph. A vital task here is to extract relational features for every node which consider not only the attributes of the node but also the attributes and labels of its partially labeled neighborhood. The Collective Classification Task is a classical Semi-Supervised Learning (SSL) problem where node representations are obtained by leveraging the information from large amounts of

unlabeled nodes in addition to the information from the labeled nodes. It is often the case that a node is not only influenced by its immediate neighbors but also by its higher order neighbors,multiple hops away which may be both labeled and unlabeled. Graph Classification is a semi-supervised learning problem where we try to classify the entire graph based on a single representation that we generate using the node and well as structural information.

Graph-structured data can appear in many real-world and scientific fields, e.g., knowledge graphs, recommender systems, social and citation networks, as well as telecommunication and biological networks . In general, a graph can be viewed as a network of nodes and edges, where nodes correspond to individual objects,and edges encode relationships among those objects. For example, in an online forum, each discussion thread can be constructed as a graph where nodes represent users, and edges represent commenting activities between users . Early approaches aim to compute similarities among graphs to build a graph kernel for graph classification . These graph kernel-based approaches split a given graph into"atomic substructures" (e.g., subtree structures, random walks, or shortest paths) to constructa frequency vector to represent the entire graph; hence they ignore node features.Recently, graph neural network (GNN)-based approaches become an essential strand to learn low-dimensional continuous embeddings of the entire graphs to predict graph labels. These GNN-based supervised approaches usually consist of two typical phases: the propagating phase and the readout phase . The former phase aims to update the vector representation of each node by recursively aggregating representations of its neighbors. Then the latter stage applies a pooling function (i.e., a READOUT operation, e.g., sum, mean, or max pooling) on output node representations to produce an embedding for each entire graph. Finally, this graph embedding is used to train a classifier to predict the graph label. We find that these supervised approaches are showing promising performances. Nonetheless, the dependency aspect among nodes, which often exhibit strongly in many kinds of real-world networks, has not been exploited effectively because of the lack of advanced computations within the propagating phase.Most of the existing approaches have focused on the supervised setting where they use the graph labels during the training process . In a situation where no graph labels are available during training, some work has considered the unsupervised setting ;however, they produce lower performances compared with the supervised approaches. Be-sides, as mentioned earlier, node features were not utilized

in these unsupervised approaches even though, except for the graph labels,they have access to all graph and node information from the entire dataset. The transformer self-attention network has successfully utilized in natural language processing tasks such as question answering, machine translation,and language modeling.

1.3 Contribution of this Thesis

In this thesis, we present Multi-pooling Convolution operation. Queuing from the world of image and text analysis, we draw parallels between this and multi-head self attention and show how this is a stricter form of multi-head attention and how the same concepts of attention maps can be used for interpretability of these networks. We also extensively test the performance of this network on different datasets and hyperparameter settings and provide a comprehensive study for the same.

CHAPTER 2

Background and Related Work

Network data have inherent relational structure among the nodes of a graph. The relational structure (edges) among nodes might encode additional latent correlated information which might be useful. For example, in most chemical datasets, the edges . Such relational information is deemed useful if it gives additional evidence that might improve the belief obtained from using only the note features. Such benefits might arise when the node features are not sufficient or robust as a stand-alone information for the end task. Representations for the graph are learned at node or edge or graph level depending on the end task. Herein, we focus on learning representations at node (vertex) level for the task of classifying nodes. Classifying nodes in Information networks such as social networks, citation networks, protein interaction networks, etc. involve characterizing the nodes' information along with its neighborhood information concerning their relational structure, attributes and their complex non-linear correlation with the labels. Early popular approaches aim to decompose each graph into "atomic substructures" (e.g.,graphlets, sub-tree structures, random walks, or shortest paths) to measure the similarity between two graphs (G artner et al., 2003). For this reason, we can view each atomic sub-structure as a word token and each graph as a text document. Hence we represent a collection of graphs as a document-term matrix that describes the normalized frequency of terms in documents. Then, we can use an inner product to compute similarities among graphs to derive a "kernel matrix" that we can use to measure the classification performance using a kernel-based learning algorithm such as Support Vector Machines (SVM) .Since the introduction of word embedding models, i.e., Word2Vec and Doc2Vec , several works have used them for the graph classification task. Deep Graph Kernel (DGK) applies Word2Vec to learn embeddings of atomic substructures to create the kernel matrix. Graph2Vec employs Doc2Vec to obtain embeddings of entire graphs to train an SVM classifier to perform classification. Anonymous Walk Embedding (AWE) maps random walks into "anonymous walks"; AWE also views each anonymous walk as a word token, and then utilizes Doc2Vec to achieve the graph embeddings to construct the kernel matrix. Recent works have focused on

using deep neural networks to perform the graph classification in the supervised setting. These are analogous to the convolution neural network architectures used in most image processing pipelines.

In general, graph neural networks (GNNs) aim to update the vector representation of each node by recursively propagating the vector representations of its neighbors using a recurrent aggregation function until convergence . The aggregation function can be a neural network, e.g., gated recurrent unit (GRU) or MLP. Multiple stacked GCN, GraphSAGE, and GAT layers are also variants of this function in GNNs. Other graph embedding models are summarized in .

2.1 Graph Based Semi - Supervised Learning

Semi-Supervised Learning (SSL) leverages unlabeled data to achieve improved generalization on supervised tasks. Typically, it is useful when the supervision is limited. In this work, we focus on the standard SSL setup where the supervision is provided in the form of target labels for data instances. Semi-Supervised Learning aims at improving the performance of the supervised learning task by obtaining a better estimate of the underlying data distribution with the available additional unlabeled data. Similar to the necessary assumptions that are made for supervised learning such as continuity, independent and identically distributed data(IID), etc., there are necessary assumptions required for SSL to work. It is vital to ensure that these assumptions, specifically those on the data, are satisfied when coupled with models that jointly learn representations for data. The popular graph-based SSL paradigm that is built on the manifold assumption preserves the geometry of data especially under the change of representation. Graph-based SSL methods model the underlying data manifold as a graph and utilize it to enforce smoothness on a target space. The graph can either be computed from the data or given apriori. Both labeled and unlabeled data are represented as nodes, and the edges denote a notion of similarity. When the graph is not given apriori, the weights on the edges are computed with some similarity kernel on the input space. The required smoothness properties are encoded with an appropriate spectral (graph) kernel.

2.2 Spectral Kernels

The different graph-based learning approaches for the Collective Classification task canbe broadly classified into two paradigms based on the use of spectral kernels:

- Graph-based regularization with kernels as similarity functions.
- Graph feature extraction with kernels as filters.

In a node classification task, the kernels operate in the vertex domain, and hence the regularization and feature extraction are over the neighborhood of vertices. Regularization based methods enforce nodes be similar to their neighbors in the label or embedding space, and feature extraction methods aggregate neighborhood information and combine it with the node features.

2.2.1 Regularization with Spectral Kernels

Let L denote the normalized symmetric Laplacian operator, where $L = I - D^{-1/2}AD^{-1/2}$ with I being the Identity matrix. The Laplacian operators defined on undirected graphs are symmetric matrices. This property of Laplacian matrices is the foundation for the spectral theory. As graph Laplacian L is a real symmetric matrix, it can be diagonalized to obtain orthonormal eigenvectors U with associated real and non-negative eigenvalues Λ. The multiplicity of eigenvalue 0, corresponds to the number of connected components in the graph. The eigenvectors corresponding to smaller eigenvalues of L encode smooth variations for connected nodes i.e $|Ui(m)Ui(n)|$ is small.The Laplacian operator on a function of graph signal, $f = f(X)$ penalizes the change between adjacent vertices as described in 2.8. Following from the definition of a Positive Semi-Definite (PSD) matrix, the graph Laplacian operator can be used as a semi-norm on signals making it a useful tool for regularization.

$$< f, Lf >= f^T Lf = \sum_{(i,j)\in E} \|f_i - f_j\|^2 \geq 0 \tag{2.1}$$

We can define a class of regularization functionals, $r(\theta)$ on the graph by parameterizing a function over the the Laplacian which would be a function over the eigenvalues as given in Eqn: 2.10. The choice of function should be driven with the thought that we

need to penalize V_i which has large Λ_i, hence the r_Λ should monotonically increase with Λ.

$$L = U^T \Lambda U \tag{2.2}$$

$$r(L) = r(\Lambda) = U^T r(\Lambda) U \tag{2.3}$$

The most common choices of graph regularizers , r are the plain Laplacian 2.11 and the regularized Laplacian 2.12 with $\sigma = 1$ followed by 2.13.

$$r(\Lambda) = I\Lambda \tag{2.4}$$

$$r(\Lambda) = \left(I + \sigma^2 \Lambda\right) \tag{2.5}$$

$$r(\Lambda) = \exp\left(\sigma^2/2L\right) \tag{2.6}$$

As long as $r(L)$ is a PSD matrix, we can define a Reproducing Hilbert space with kernel, $= r(L) - 1$.

Laplacian smoothing

These Regularization kernels are used to encode smoothening priors based on the relational structure of G. Laplacian regularizers have been extensively used for semi-supervised node classification. Below, we provide a functional definition of Laplacian regularizer parameterized by the graph of consideration and the space to be smoothed.

$$LS(P, f(X)) = \sum_{(i,j)\in E} \|f(X_i) - f(X_j)\|^2 = f^T \Delta(P) f \tag{2.7}$$

Let Δ denote the graph Laplace operator. Let $LS(P, f(X))$ denote Laplacian Smoothing (LS) on the target space, $f(X) with L = (P)$, where $P \epsilon R^{nn}$ is some defined proximity matrix. As mentioned earlier, Laplacian smoothing can be applied on the label space $f(X) = L$, data projections X or $f(X)$ or on a latent cluster space, $f(X) = C$ as proposed in this work.

$$Loss = Loss_{Supervised} + \lambda LS(P, f(X)) \tag{2.8}$$

The regularization term is explicitly added to learning objective and is optimized along with the supervised loss as in 2.15. The λ in the equation is a trade-off parameter which weighs neighborhood similarity with the node features. λ is chosen by cross-validation to determine the importance of node features. Laplacian SVM is a popular model which couples a Laplacian regularizer to the Hinge loss and one can also view Label propagation as Laplacian smoothening on the label space with a k-nearest neighbor classifier.

2.2.2 Convolutions with Spectral Kernels

The spectral of the Laplacian help us form analogous theory to Fourier transforms leading to Graph Fourier Transforms enabling us to model filters which can be used for convolution operation in the original graph space as in Eqn: 2.16. Unlike kernels for regularization which focused on monotonic scaling of the eigenvalues, kernels for filtering are chosen to parameterize the combination of the laplacian basis.

$$g_\theta * x = U g_\theta(\Lambda) U^T x \tag{2.9}$$

The non-parametric kernels as in Eqn: 2.17 are potent alternatives to parametric filters as they are not biased to any specific family of spectral transformations and also have higher degrees of freedom of the order of number of nodes in comparison to parameterized kernels with a fixed, K filters . However, they are less desirable as the number of parameters grows with growing vertex set, and also these filters (parameters) are not localized. The popular alternative is the polynomial filter given in Eqn: 2.18. The polynomial filters of the K-th order are exactly K-localized. However, extracting features with these polynomial filters are compute intensive owing to the dense-dense multiplication of the signal, x and the basis, U besides the running time complexity of the eigen decomposition $O(n^3)$.

$$\text{Non - parametric filter : } \; g_0(\Lambda) = diag(\theta) \tag{2.10}$$

$$\text{Polynomial filter : } \; g_\theta(\Lambda) = \sum_{k=0}^{K} \theta_k \Lambda^k \tag{2.11}$$

In order to overcome this computationally expensive step, we can resort to approximate computations of polynomials such as that of Chebyshev polynomial approximations. A truncated series of Chebyshev polynomials, which can be defined recursively in terms of L(Eqn: 2.19) is used to approximate the Polynomial filter i.e we intend to approximate $g_o(L)$ directly over $g_o(\Lambda)$ as given in Eqn: 2.19 as seen in Eqn: 2.20. This avoids the need to compute the full set of eigen-vectors and more importantly, we can now leverage sparse-dense multiplications to compute Lx as L is a sparse matrix. This reduced the complexity to the $O(K|E|)$.

$$\begin{aligned} T_0(x) &= x \\ T_1(x) &= L^- x \\ T_k(x) &= 2L^- T_{k-1}(x) - T_{k-2}(x) \end{aligned} \tag{2.12}$$

$$\begin{aligned} \sum_{k=0} \theta_k \Lambda^k &\approx \sum_{k=0}^{K} \theta_k T_k\left(L^-\right) x \\ \text{where } L^- &= \frac{2}{\lambda_{\max}} L - I_N \end{aligned} \tag{2.13}$$

2.2.3 Chebyshev Neural Network

The Chebyshev neural network of is built by stacking multi-ple K^{th} order Chebyshev filters mentioned in Eqn: 2.20. Chebyshev Neural networks defined in Eqn: 2.21 was defined for extracting graph level features to make graph classification. Graph Coarsening layers are interleaved between Chebyshev layers to coarsen the graph iteratively, and graph pooling layer was defined to summarize information from all the nodes of the graph at every layer. All Chebyshev layers have ReLU as the activation function, i.e $\sigma_k = \text{ReLU} \forall k \epsilon [1, K]$. The features from the final layer are sent through an additional graph classification layer.

$$\begin{aligned} h_0 &= X \\ h_{k+1} &= \sigma_k\left(g_\theta\left(L^K\right) h_k\right) \\ &= \sigma_k\left(\sum_{k=0}^{K} \theta_k T_k\left(L^-\right) h_k\right) \end{aligned} \tag{2.14}$$

2.3 Graph Convolutional Networks

(Kipf and Welling (2016)) introduced Graph Convolutional Networks (GCNs) as stacked approximations of first-order Chebyshev filters. They are also multi-layer convolutional neural network where the convolutions are defined on a graph structure for the problem of Semi-Supervised node classification. The significant improvement over Chebyshev Neural Networks is that, in GCNs, filtering happens at every step of the propagation thereby allowing only relevant information to flow across neighborhoods. This also enables GCNs to learn complex non-linear feature basis.The conventional two-layer GCNs which captures information up to the 2^{nd} hop neighborhood of a node can be reformulated to capture information up to any arbitrary hop, K as given below in Eqn: 2.22.

$$\begin{aligned} h_0 &= X \\ h_k &= \sigma_k\left(\hat{L}h_{k-1}W_k\right), \forall k \in [1, K-1] \\ Y &= \sigma_K\left(\hat{L}h_{K-1}W_L\right) \end{aligned} \tag{2.15}$$

GCNs were used for multi-class classification task with Rectified Linear Unit (ReLU)as the activation function, i.e σ_k = ReLU$\forall k[1, K1]$ and a softmax label layer, $\sigma_k = softmax$. We can rewrite the GCN model in terms of $(K1)^{th}$ hop node and neighbor features as below by factoring $\hat{L}$.

$$\begin{aligned} h_k &= \sigma_k\left(\left(\hat{D}^{-\frac{1}{2}}I\hat{D}^{-\frac{1}{2}} + \hat{D}^{-\frac{1}{2}}A\hat{D}^{-\frac{1}{2}}\right)h_{k-1}W_k\right) \\ &= \sigma_k\left(SUM\left(\hat{D}^{-1}h_{k-1}, \hat{D}^{-\frac{1}{2}}\Lambda\hat{D}^{-\frac{1}{2}}h_{k-1}\right)W_k\right) \end{aligned} \tag{2.16}$$

2.3.1 Relation to Weisfeiler-Lehman (WL) algorithm

GCNs can be viewed as continuous approximations of the node features obtained with 1-dim Weisfeiler-Lehman algorithm (Kipf and Welling (2016); Hamiltonet al.(2017)).The Weisfeiler-Lehman (WL) algorithm defines a vertex color (label) refinement scheme that is used to obtain graph signatures to perform graph-isomorphism tests. The WL algorithm starts with an initial coloring (discrete features), for all nodes and uses a hashing function that aggregates neighbors' colors and produces a new color. This step is repeated multiple times until we get a stable coloring. Eqn: 2.24 computes the color of a node, i in k^{th} iteration based on a hashing of neighbors' colors from the $k-1^{th}$

step.

$$\text{color}_k[i] = \text{hash}\left(\sum_{(i,j)\in E} \text{color}_{k-1}[j]\right) \tag{2.17}$$

The WL-algorithm, at every step, k applies a local function to compute k-hop signatures/representations of nodes. At every step, the information from a node gets propagated further away as shown in Figure: 2.1. Figure: 2.1 shows information at the nodes at different step, with leftmost being the initial 0^{th} step and the rightmost being 2^{nd} step. At every step, each node aggregates neighbors symbols and takes a union of it. GCN models are parameterized differentiable extensions of WL algorithm based

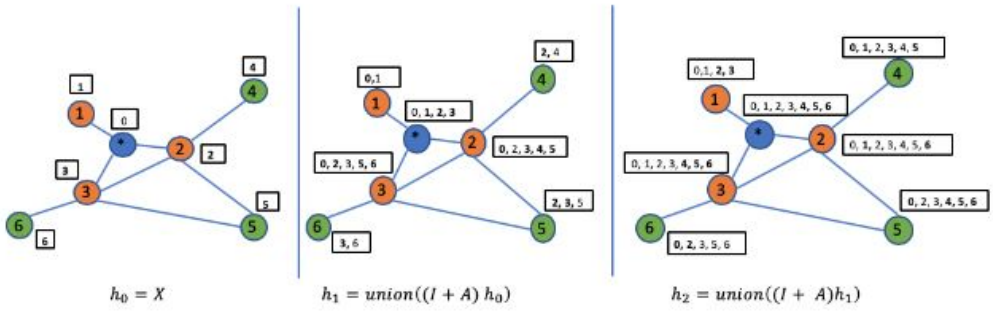


Figure 2.1: Information Propagation in 1-D W-L Kernel

graph kernels, where the hashing function is replaced with a non-linear function, and the simple union aggregation function is replaced with differentiable functions such as mean with GCNs.

2.4 GraphSAGE

Graph SAmple and AggreGatE model (GraphSAGE) proposed in (Hamiltonet al.(2017)) consists of 3 models with different differentiable neighborhood aggregator functions. GraphSAGE models were defined for the multi-label Semi-Supervised inductive learning task, i.e generalizing to unseen nodes during training. Let the function $Aggregate()$ abstractly denote the different aggregator functions in GraphSAGE, specifically $Aggregate[mean, maxpooling, LSTM]$ and we will refer to these models as GS-MEAN, GS-MAX and GS-LSTM, respectively. Similar to GCN, GraphSAGE models also re-cursively combine neighborhood information at each layer of the Neural Net-

work. GraphSAGE has an additional label layer unlike GCN, i.e. ,$h_L = h_{K+1}$. Hence, $\sigma_k = \text{ReLU} \forall k \epsilon [1, K]$ and $\sigma_L = \text{sigmoid}$. Here, the weights $W_{\hat{k}} \epsilon R^{2dxd}$.

$$\begin{aligned} h_0 &= X \\ h_k &= \sigma_k \left(\text{ CONCAT } (h_{k-1},\ \text{ Aggregate } (h_{k-1}, A)\, W_{\hat{k}}) \right. \\ &\quad \forall k \in [1, K] \\ Y &= \sigma_{K+1} (h_K W_L) \end{aligned} \tag{2.18}$$

GraphSAGE models, unlike GCN, are defined to work with partial neighborhood information. For each node, these models randomly sample and use only a subset of neighbors from different hops. This choice to work with partial neighborhood information allows them to scale to large graphs but restricts them from capturing the complete neighborhood information. Rather than viewing it as a choice it can also be seen as restriction imposed by the use of Max Pool and LSTM aggregator functions which require fixed input lengths to compute efficiently. Hence, GraphSAGE constraints the neighborhood sub-graph of a node to contain a fixed number of neighbors at each hop.

2.5 Graph Attention Network

The practical importance of attention in deep learning is well-established and there are many argu-ments in its favor [1], including interpretability [2,3]. In graph neural networks (GNNs), attention can be defined over edges [4,5] or over nodes [6]. Graph Attention operator is an operator that applies attention over nodes. In order to convert input features into higher level features, at least one learnable linear transformation is required. Hence, the initial step is a shared linear transformation, parameterized by a weight matrix, $W \epsilon \text{R}^{2f(n+1)\text{x}f(n)}$ (where f_i is the feature length in the i^{th} layer), applied to each node. A shared attention mechanism $a : R^{2f(n+1)} \longrightarrow R$ computes the attention coefficients.

$$e_{ij} = a\,(Wh_i, Wh_j)\, e_{ij} \text{ is importance of node } j's \text{ features to node } i's \tag{2.19}$$

To constrict these attention values between 0 and 1, we apply softmax to the attention values.

$$\alpha_{ij} = \frac{\exp\left(e_{ij}\right)}{\Sigma_{k \in N_i} \exp\left(e_{ik}\right)} \tag{2.20}$$

N_i = all the neighbours of node i Specifically in the paper:

$$\alpha_{ij} = \frac{\exp\left(\text{leaky ReL}U\left(a^T\left[Wh_i \| Wh_j\right]\right)\right)}{\sum_{k \in N_i} \exp\left(\text{leaky Re}LU\left(a^T\left[Wh_i \| Wh_k\right]\right)\right)} \tag{2.21}$$

Hence, the final embedding of node i is given as a weighted sum of the transformed embeddings of itself as well as all its neighbours. Weights being the attention values:

$$h_{i+1} = \Sigma_{k \in N_i} \alpha_{ij} \text{ leakyRelu}\left(Wh_k\right) \tag{2.22}$$

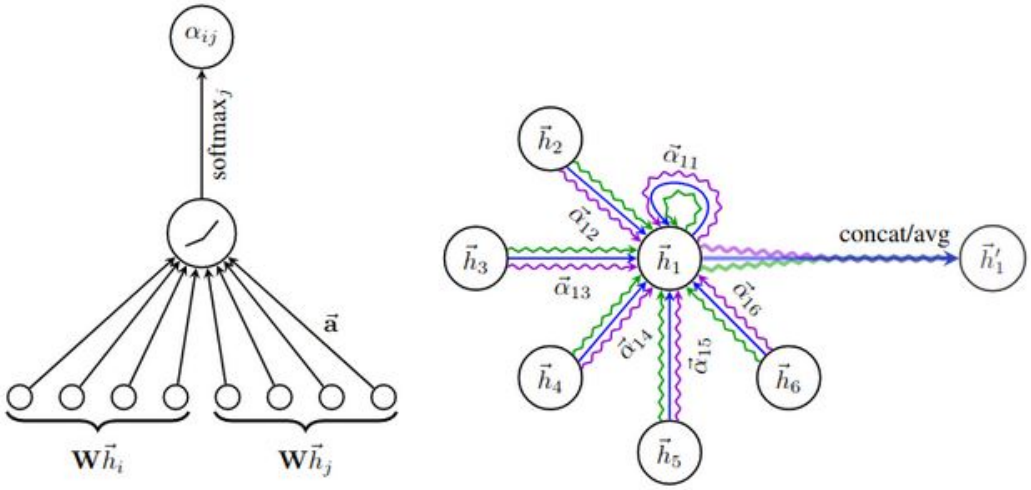


Figure 2.2: **Left:** The attention mechanism employed in GAT. **Right:** An illustration of multi-head attention (with K= 3heads) by node 1 on its neighborhood. Different arrow styles and colors denote independent attention computations. The aggregated features from each head are concatenated or averaged to obtain the final embedding h'_1

Here, instead of leakyReLU, you can also use any other non-linear operator such as tanh or elu

2.6 Global Attention

Global attention is essentially a Graph Attention Operator applied to a fictitious node which is connected to all nodes. It is a global pooling Operation to generate a single representation of the entire graph. Hence global attention can be written as:
GlobalAttention(G) = GAT($Node_G$), such that
$h_G = \phi$ and $N_G = N$

2.7 Message Passing Neural Networks

All graph convolution based frameworks ,which do not incorporate edge features, could be reformulated into a single framework known as Message Passing Neural Networks. Message Passing Neural Networks (MPNNs) are a term coined by Google research scientists working on neural computation for predicting quantum states of molecules. MPNNs broadly describe unstructured graph-networks, where the nodes are neurons and the edges are like a synapse between two neurons with a particular (and modifiable) strength. When the connection strengths are updated in a rule-based manner until they converge, these graph-based networks can be used to perform complex computation with very high data throughout. MPNN framework can be divided into two parts:

2.7.1 Message Phase

Where information is aggregated and nodes are embedded:

$$m_v^{t+1} = \Sigma_{w \in N(v)} M_t \left(h_v^t, h_w^t, e_{vw}\right) \tag{2.23}$$

$$h_v^{t+1} = U_t \left(h_v^t, m_v^{t+1}\right) \tag{2.24}$$

2.7.2 Read - Out Phase

Where the embeddings are used to find a representation for the entire graph.

$$y^* = R\left(\left\{h_v^t | v \in G\right\}\right) \tag{2.25}$$

Here:
M_t is a message function which generates the message m_v^{t+1} based on all the neighbours. This message along with the previous embedding of the node is used to generate the updated embeddings. y^* is a single vector representation generated for the entire graph based on the generated from the node embeddings. R must be invariant to permutation of nodes for the network to be invariant to graph isomorphism. Multiple readout phase layers can be stacked on top of one another to build deeper networks. This is a generalised framework for all deep learning models.

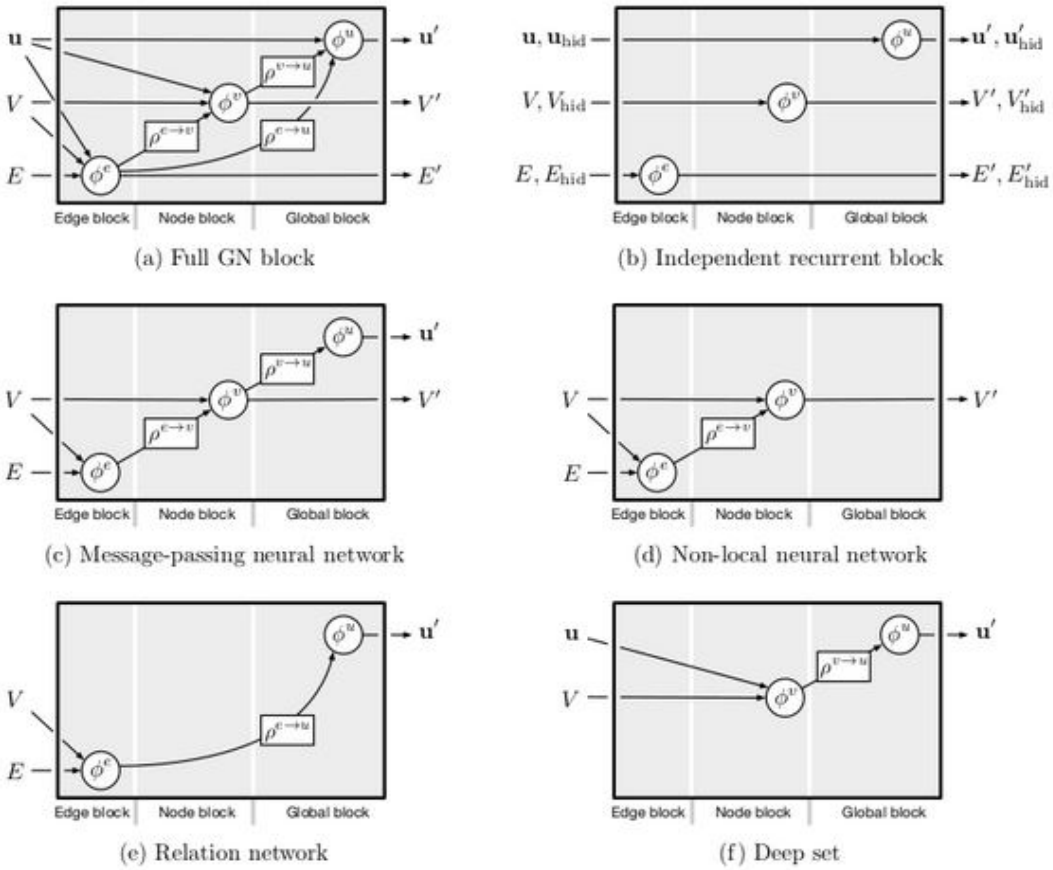
2.8 GraphNets

Finally Graph Nets were developed which also update edge information after every iteration as well as incorporation of global representation of the entire graph to update the individual node representations. In graph nets, a graph is not just defined by vertex set, V, and edge set, E, but by $u \epsilon R^f$ which is a global representation of the entire graph.

Almost all frameworks can be written in the Graph Nets framework and this extends to all graph types and graph tasks. Hence it is the most comprehensive Graph Convolution architecture.

Algorithm 1 Steps of computation in a full GN block.

function GRAPHNETWORK(E, V, $\mathbf{u}$)
  for $k \in \{1 \dots N^e\}$ **do**
    $\mathbf{e}'_k \leftarrow \phi^e\left(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}, \mathbf{u}\right)$ ▷ 1. Compute updated edge attributes
  end for
  for $i \in \{1 \dots N^n\}$ **do**
    let $E'_i = \left\{\left(\mathbf{e}'_k, r_k, s_k\right)\right\}_{r_k=i,\, k=1:N^e}$
    $\bar{\mathbf{e}}'_i \leftarrow \rho^{e\to v}\left(E'_i\right)$ ▷ 2. Aggregate edge attributes per node
    $\mathbf{v}'_i \leftarrow \phi^v\left(\bar{\mathbf{e}}'_i, \mathbf{v}_i, \mathbf{u}\right)$ ▷ 3. Compute updated node attributes
  end for
  let $V' = \left\{\mathbf{v}'\right\}_{i=1:N^v}$
  let $E' = \left\{\left(\mathbf{e}'_k, r_k, s_k\right)\right\}_{k=1:N^e}$
  $\bar{\mathbf{e}}' \leftarrow \rho^{e\to u}\left(E'\right)$ ▷ 4. Aggregate edge attributes globally
  $\bar{\mathbf{v}}' \leftarrow \rho^{v\to u}\left(V'\right)$ ▷ 5. Aggregate node attributes globally
  $\mathbf{u}' \leftarrow \phi^u\left(\bar{\mathbf{e}}', \bar{\mathbf{v}}', \mathbf{u}\right)$ ▷ 6. Compute updated global attribute
  return $\left(E', V', \mathbf{u}'\right)$
end function



(a) Full GN block (b) Independent recurrent block

(c) Message-passing neural network (d) Non-local neural network

(e) Relation network (f) Deep set

Figure 2.3: The figures above show how a Graph Net block encompasses all graph convolutional algorithm and even Recurrent Neural Networks. Changing the internal block connections and altering block computations helps achieve this.

2.9 Graph Pooling

Most of the research in Graph Representation learning, specifically for graph classification is based on generalising graph convolutional neural network architectures to graphs. This also extends to pooling methods, which have proven to be a challenging task given the irregular structure of the graphs. Defining pooling operation in graphs is not as simple as in images, since the structures are irregular and hence symmetric division of nodes is not possible. Graph pooling algorithms can also be seen as an extension to graph coarsening to incorporate node information as well. Most popular based pooling algorithms can broadly be divided into two types, one that incorporates clustering to generate completely new nodes and the other which uses a thresholding function to filter out nodes. Here we will explain three of the most popular pooling methods: Spectral Clustering, Differential Pooling, Top-K pooling and Self-Attention Graph Pooling.

2.9.1 Spectral Clustering

If you consider the eigenvector entries of L to be some labels given to each node, then u1 (vector corresponding to the smallest eigenvalue) is giving all the nodes the same label. U2 divides the nodes into two clusters- one with a value greater than 0 and the others whose value is less than zero. Eigenvectors associated with larger eigenvalues have values that vary more rapidly along the edges. There are two ways to partition graphs using this:

- Recursive Bi-partitioning, or
- Spectral Clustering.

The biggest issue with method is, pooling via this method is based solely on the graph structure. Also calculating eigen-decomposition for large graphs can be computationally expensive

2.9.2 Differential Pooling

Here, for every pooling operating we use a learnable cluster assignment matrix $S^{(l)} \epsilon R R^{n_l x n_{l+1}}$ to generate a new graph using the node embeddings at that layer: Hence this cluster assignment matrix is used not only to generate new nodes, but also to build the new adjacency matrix. Hence differential pooling leads to complete graph reconstruction.

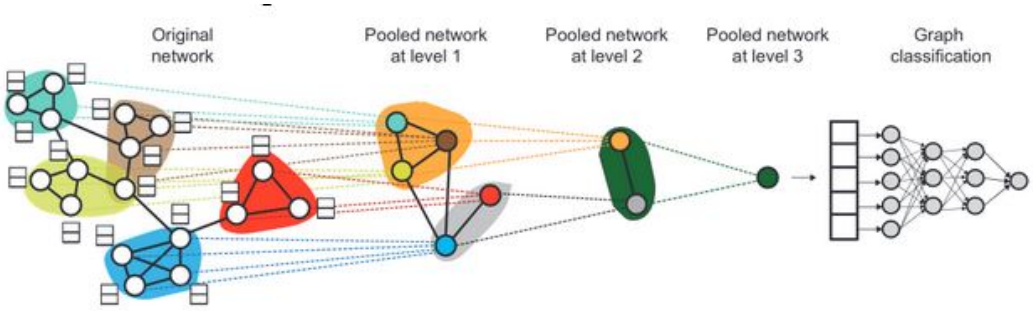


Figure 2.4: High-level illustration of DIFFPOOL algorithm applied recursively along with Graph Neural Network Operators to Coarsen the graph while retaining information. At each hierarchical layer, a GNN model is used to obtain embeddings of nodes. These learned embeddings are then used to cluster nodes together and run another GNN layer on this coarsened graph. This whole process is repeated for L layers and the final output representation is used to classify the graph

$$X^{(l+1)} = S^{(l)^T} Z^{(l)} \in R^{n_{l+1} \times d} \tag{2.26}$$

$$A^{(l+1)} = S^{(l)^T} A^{(l)} S^{(l)} \in R^{n_{l+1} \times n_{l+1}} \tag{2.27}$$

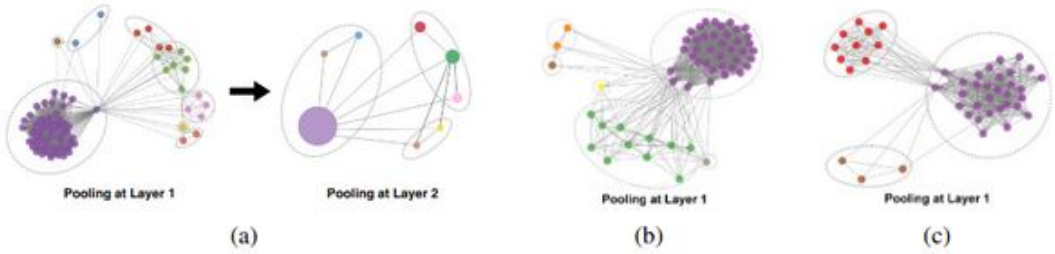


Figure 2.5: Visualization of hierarchical cluster assignment in DIFFPOOL for different graphs

2.9.3 TopK Pooling

TopK pooling is a pooling method wherein nodes are filtered based on a score. This score is generated by projecting the node features onto a learnable vector p. The number of nodes to filter can either be set by filtering a fixed fraction of the original number of nodes or by setting a fixed threshold value.

$$\begin{aligned} Y' &= X^i p / \|p\| \\ ldx &= \text{rank}\,(y', k) \\ X^{i+1} &= X'(idx, \quad) \\ A^{i+1} &= A^i(idx, idx) \end{aligned} \tag{2.28}$$

where,

X^i = Node embeddings at layer l

A^i = Adajency matrix at layer l

X^{i+1} = Node embeddings at layer l+1

A^{i+1} = Adajency matrix at layer l+1

K = Number of nodes selected in the new graph

rank(y,k) = Operation of node ranking, which returns indices of the k-largest values in y

2.9.4 Self Attention Graph Pooling

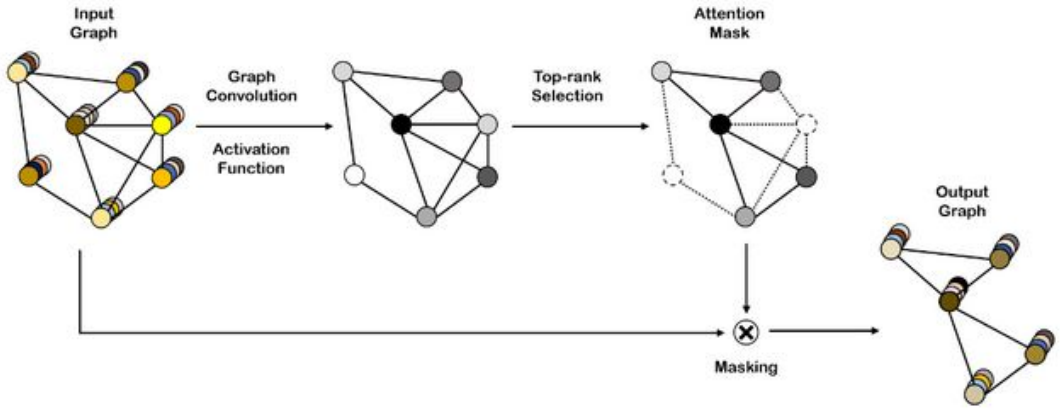


Figure 2.6: An illustration of the SAGPool layer.

SAGE pooling employs a similar method to TopK pooling, but it uses a Graph Neural Network, rather than projection onto a learnable vector, to generate the threshold

scores.

$$Z = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}X\Theta_{att}\right) \tag{2.29}$$

$$\text{idx} = \text{top} - \text{rank}(Z, \lceil kN \rceil), \quad Z_{\text{mask}} = Z_{\text{idx}} \tag{2.30}$$

$$X' = X_{\text{idx},:}, \quad X_{out} = X' \odot Z_{mask}, A_{out} = A_{\text{idx,idx}} \tag{2.31}$$

CHAPTER 3

Multi Pooling Convolution

Rather than taking all embeddings during convolutions, we can break the graph into several sub-graphs, perform embeddings of these subgraphs individually and then combine the entire graph together. This can be seen analogous to multi-head attention used in most modern text analysis models. Though the diagram below suggests use of Graph

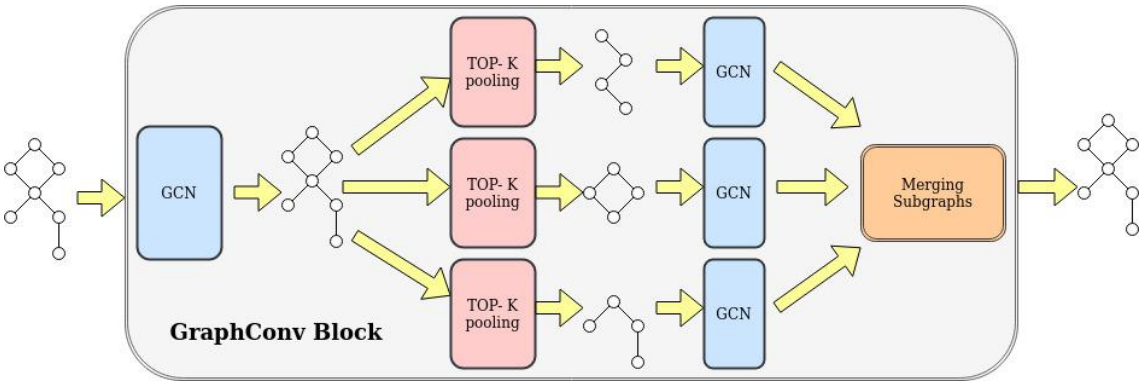


Figure 3.1: Graph Convolution Block proposed by us to perform convolution operation. The number of sub-graphs to generate, $\text{num}_h eads, isahyper - parameterthatneedstobetunedbytheuser.Acomprehensivestudyonthisisprovidedlate$

Convolution operator (GCN), we will test the network with other types of convolutions as well (GraphSage, Graph Isomorphism Operator, ChebyShev Convolution Operator, etc). To remove redundancy in selecting sub-structures, and effective aiding in network stability, we define a new network specific loss function which penalizes the model if it selects similar sub-structures.

Subgraph Similarity Loss : $\alpha \sum_i^n \sum_j^n \langle p_i, p_j \rangle / \|p_i\| \|p_j\|$, where

n = number of subgraphs generated in that convolution block

p_i = the trainable vector used to generate the ith sub-graph

α= hyperparameter to be tuned

Many such blocks can be stacked on top of one another to form a deep network. Similar to depth, the number of subgraphs to extract ,i.e., width of the network can also be altered.

The above network can be seen as a graphical (,i.e., with irregular geometry, an image can be seen as a graph with grid geometry) adaption of a CNN (Convolutional

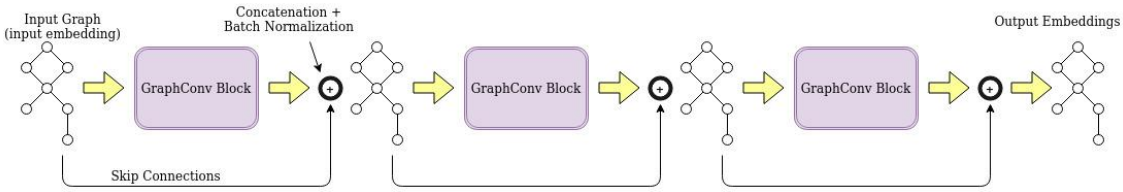


Figure 3.2: Multiple such blocks can be stacked together along with residual connections to build a deep network

Neural Network), specifically Residual Network. Consequently the analysis methods used in CNNs, such as filter analysis, can be used to study the substructures deemed important by the Network. Hence this network is more important from an interpretability viewpoint to aid the study of chemical and physical properties of a compound/lattice structure, or to view the salient substructures in a point-cloud or other such geometric .

The above network provides node level embeddings and can be used only for collective classification tasks. Some global pooling operations need to be performed to generate a single representation for the entire graph. You could use use add or max pooling or can even employ global attention pooling on the entire network. To this end, specifically for the task of classification, we have built a pooling block which, as the GraphConvBlock, breaks the graph into several subgraph, but here, instead of merging them back, it uses global attention on each of these subgraphs to get a single representation and then uses these subgraphs to get a class specific score. Hence in this final layer, there will be as many sug-graphs generated as there will be total classes in the dataset.

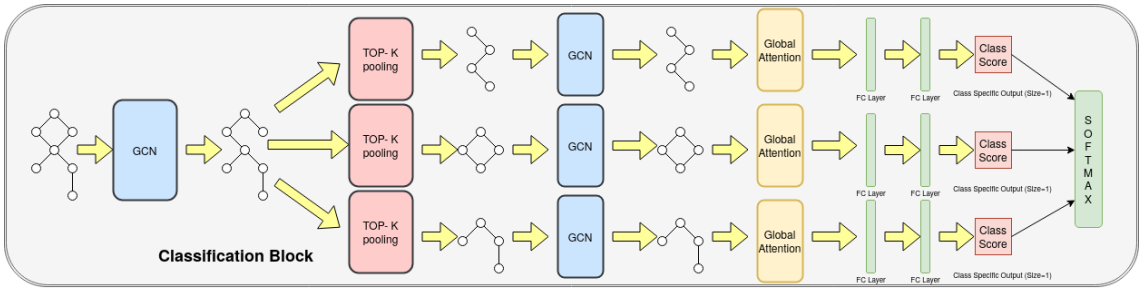


Figure 3.3: The Classification proposed by us. Here we break the graph into multiple sub-graphs, similar to Graph Convolution Block defined earlier, and perform global attention on each of these sub-graphs to generate a single representation of the entire sub-graph. These sub-graph representations will then be used to generate class-specific scores.

Hence, outputs from these sub-graphs can be seen as class specific scores and a softmax layer can be applied on top of this to get the final classification output. Such a

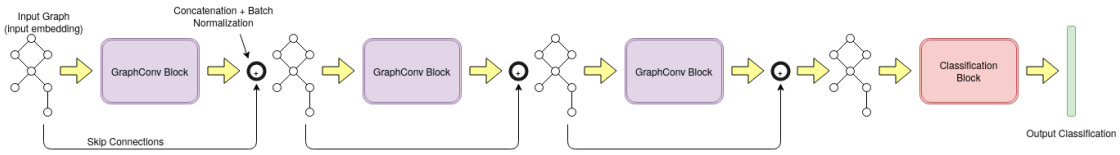


Figure 3.4: The final network architecture for graph classification task. The number f blocks to stack is a hyper-parameter that needs to be set by the user. The number of blocks does not include the final classification block.

structure is also useful as it can be used for interpretation as to which type of graphs or which specific traits in a graph give a high output in a particular type of graph.

CHAPTER 4

Dataset Description

For the testing of this network, we specifically work with three types of datasets:

- Datasets with node embeddings and a single edge type. Usually chemical and biomedical datasets.
- Datasets with neither node nor edge embeddings. These are purely topological datasets. Usually social media datasets.
- Node Classification with a single large graph. These are either citation networks or social media datasets.

Datasets with single edge type (,i.e., no edge classes, ede embeddings):

1. PROTEINS full: are sets of proteins from Dobson and Doig (2003) database. Vertices are secondary structure elements as in [Borgwardtet al., 2005] and have three distinct labels, representing helix,sheet or turn. PROTEINS is a dataset obtained from where nodes are secondary structure elements (SSEs) and there's an edge between two nodes if they are neighbors in the amino-acid sequence or in 3D space. The aim is to predict the function of a particular protein.
2. ENZYMES : are sets of proteins from the BRENDA database[Schomburget al., 2004]. Vertices are secondary structure elements as in[Borgwardtet al., 2005]. ENZYMES is a balanced dataset of 600 protein tertiary structures obtained from and has 3 discrete labels. The aim is to classify the type of reaction the particular enzyme catalyzes.
3. MUTAG : MUTAG is a dataset of 188 mutagenic aromatic and heteroaromatic nitro compounds with 7 discrete labels.
4. DD: DD is a dataset of 1178 protein structures (Dobson and Doig, 2003). Each protein is represented by a graph, in which the nodes are amino acids and two nodes are connected by an edge if they are less than 6 Angstroms apart. The prediction task is to classify the protein structures into enzymes and non-enzymes.
5. COLORS-3:Synthetic graph dataset where features for each node are assigned to one of the three one-hot values (colors): [1,0,0] (red), [0,1,0] (green),[0,0,1] (blue). The task is to count the number of green nodes, Ngreen.
6. FRANKENSTEIN: Is a dataset created by the fusion of the BURSI and MNIST datasets. BURSI is made by 4337 molecules with mutagenicity (AMES) classification, with 2401 mutagens and 1936 non-mutagens[Kaziuset al., 2005]. Each molecule is represented as a graph whose vertices are labeled by the chemical

atom symbol and edges by the bond type. FRANKENSTEIN is a modified version of the BURSI dataset: we discarded bond type information and remapped the most frequent atom symbols (vertex labels) to MNISTdigitimages. The original atom symbols can only be recovered through the high dimensional MNIST vectors of pixel inten-sities, in this sense this is a challenging problem for a graph kernel that can handle continuous attributes.

7. NCI datasets: NCI1 and NCI109 datasets (4100 and 4127 nodes, respectively), made publicly available by the National Cancer Institute (NCI) are two subsets of balanced datasets of chemical compounds screened for ability to suppress or inhibit the growth of a panel of human tumor cell lines,having 37 and 38 discrete labels respectively

No edge or node features. Purely topological datasets:

1. COLLAB: COLLAB is a scientific-collaboration dataset, derived from 3 public collaboration datasets , namely, High Energy Physics, Condensed Matter Physics and AstroPhysics. Following the approach of, we generated ego-networks of different researchers from each field, and labeled each graph as the field of the researcher. The task is then to determine whether the ego-collaboration graph of a researcher belongs to High Energy, Condensed Matter or Astro Physics field.

2. TRIANGLES: Counting the number of triangles in a graph is a well-known task which can be solved analytically by computing trace, (A3)/6, where A is an adjacency matrix. This task turned out to be hard for GNNs, so a node degree feature is added to every node (as a one-hot vector) to all graphs, so that the model can exploit both graph structure and features.

3. IMDB-BINARY: IMDB-BINARYis a movie-collaboration dataset where we collected actor/actress and genre information of different movies on IMDB. For each graph, nodes represent actors/actresses and there is an edge between them if they appear in the same movie. We gen-erated collaboration graphs onAction and Romance genres and derived ego-networks for each actor/actress. Note that a movie can belong to both genres at the same time, there-fore we discarded movies from the Romance genre if the movie is already included in the Action genre. Similar to COLLAB dataset, we simply labeled each ego-network with the genre graph it belongs to. The task is then simply to identify which genre an ego-network graph belongs to.

4. IMDB-MULTI: IMDB-MULTIs multi-class version of IMDB-BINARYand contains a balanced set of ego-networks derived fromComedy, Romance and Sci-Fi genres.

5. REDDIT-BINARY: REDDIT-BINARYis a balanced dataset where each graph corresponds to an online discussion thread where nodes correspond to users, and there is an edge between two nodes if at least one of them responded to an-other's comment. We crawled top submissions from four popular subreddits, namely, IAmA, AskReddit, TrollXChro-mosomes, atheism. IAmA and AskReddit are two question/answer-based subreddits and TrollXChromosomes and atheism are two discussion-based subreddits. The task is then to identify whether a given graph belongs to a question/answer-based community or a discussion-based community.

6. REDDIT-MULTI-5K: REDDIT-MULTI-5K is a balanced dataset from five different subreddits, namely,worldnews,videos,AdviceAnimals,aww and mildlyinteresting where we simply label each graph with their correspon-dent subreddit.

7. REDDIT-MULTI-12K: REDDIT-MULTI-12Kis a larger variant of REDDIT-MULTI-5K, consists of 11 different subreddits, namely, AskReddit, AdviceAnimals, atheism, aww, IAmA, mildlyinter-esting, Showerthoughts, videos, todayilearned, worldnews, TrollXChromosomes. The task in both datasets is to predict which subreddit a given discussion graph belongs to.

Node Classification Datasets :

1. Citation Networks: Cora, CiteSeer and PubMed are citation networks wherein, given the information related to a research paper and the research paper it cites, one has to predict the subject of the research paper. All three datasets are networks with a specific set of research genres.

Dataset	Dataset Statistics				
	Graphs	Num. Classes	Avg. Nodes	Avg. Edges	Node Attr
ENZYMES	600	6	32.63	62.14	18
PROTEINS_full	1113	2	39.06	72.82	29
Mutag	188	4	17.93	19.79	7
DD	1178	2	284.32	715.66	89
NCI1	4110	2	29.87	32.3	37
Frankenstein	4337	2	16.9	17.88	780
IMDB-BINARY	1000	2	19.77	96.53	-
IMDB-MULTI	1500	3	13	69.54	-
COLLAB	5000	3	74.49	2457.78	-
REDDIT-BINARY	2000	2	429.63	497.75	-
REDDIT-MULTI-5K	4999	5	508.52	594.87	-
REDDIT-MULTI-12K	11929	11	391.41	456.89	-
Cora	1	7	2708	10556	1433
Citeseer	1	6	3327	9104	3703
Pubmed	1	3	19717	88648	500

Figure 4.1: Graph Dataset Statistics

CHAPTER 5

Experiments

All tests were done with a 60-20-20 train-validation-test split. For reproducibility all non-deterministic computations were turned off and the pseudo-random number generator was initialized with a fixed seed. THe following are the hyperparameters to be set for the network:

- Similarity Loss (α): The subgraph similarity loss is set to either 0 or 0.1
- Embedding Size: This is the hidden and output embedding size. It is set to 32 for purely topological datasets with no node or edge features. For all other cases it is either set to 64 or 128
- Number of Heads (h): This is essentially the number of sub-graphs to break the graph into to perform individual convolutions. This is tested for various values from 1 all the way to 16. The model is coded such that the number of heads must be a factor of embedding size. So within every subgraph, the embeddings for every node are of size (Embedding Size / Number of Heads). This is done for computational reasons to keep the number of parameters independent of the number of heads.
- Number of Blocks (b): This is the number of convolution labers or GraphConv blocks to stack on top of one another, not including the classification block, to build the network. It is tested for various values from 1 all the way to 16.

CHAPTER 6

Results

The results, along with the specific hyperparameter settings are tabulated below.

Dataset	Model parameters				Accuracy	
	Convolution type	Number of heads	No. of Blocks	Embedding size	Ours	Current best
ENZYMES	GraphConv	2	3	128	67	78.4
PROTEINS_full	GraphConv	1	3	64	82	84.5
Mutag	GraphConv	4	3	64	97.3	95
DD	GraphConv	2	4	128	80.4	95.7
NCI1	GraphConv	4	1	128	76	87.2
Frankenstein	GraphConv	4	8	128	70	78.9
IMDB-BINARY	GraphConv	4	3	32	70	93.5
IMDB-MULTI	GraphConv	4	3	64	48.67	74.8
COLLAB	Chebyshev	4	8	32	71.1	95.6
REDDIT-BINARY	Chebyshev	4	8	32	87.7	92.4
REDDIT-MULTI-5K	Chebyshev	4	8	32	55.75	77.25
REDDIT-MULTI-12K	Chebyshev	4	8	32	44.9	47.8
Cora	GraphConv	4	4	128	83	89.5
CiteSeer	GraphConv	2	2	128	77	78.7
PubMed	GraphConv	4	3	128	82	87.8

Figure 6.1: Network Performance

The performance of the network varies across the board. While the network achieved state of the art performance for Mutag dataset, beating the current best by 2.3 percent (absolute, not relative), it performs very poorly in other datasets such DD, NCI1 and Collab. The biggest issue with the network is that, since we are performing tests on a

very large spectrum of network sizes, regularization becomes tricky and in a number of cases it over fits. Setting alpha to 0.1 did solve this issue in some cases, but not all. Tikhonov regularization, while curbing overfitting, did not improve network performance. All in all, strong network pruning methods are required to train this network properly

The network architecture is strongly correlated with the dataset, specifically with the average size of the graphs. Large graph datasets inherently need a deeper network to cover the entire graph with its receptive field. An intuitive understanding of this is that for a connected graph of size n, the maximum possible distance between a node and any other node is n and hence is directly proportional to the graph size. Hence I would need n single-hop convolutions for the embedding of that node to have incorporated the information of all the nodes in the graph.

Hence small graph datasets perform better in shallow networks while large ones on deeper networks. This is clearly visible here, where datasets like NCI1 perform fairly well with only one layer, while datasets like reddit datasets require extremely deep networks composed of multi-hop convolutional layers.

These observations are further strengthened with our deeper analysis with NCI1. Number of Heads, Number of blocks, Embedding Size and regularization parameters are hyper parameters that need to be tuned. To understand their effect, we performed experiments on NCI1 dataset with 140 different parameter combinations. In all these experiments were performed with regularization turned off (=0). Also the model is coded such that the number of heads must be a factor of embedding size. So within every subgraph, the embeddings for every node are of size (Embedding Size / Number of Heads). This is done for computational reasons to keep the number of parameters independent of the number of heads.

One of the major observations here is that, when you keep the dataset fixed, as you increase the number of heads, the network performance is inversely proportional to its depth. As the number of blocks increases, the network performs better with lesser heads. This inverse relation usually balances to give in optima as 4 heads and 3 blocks. Something we have also observed from our tests on other datasets.

The observations, although, are not very conclusive due to the results which vary a lot, making it difficult to come down to such simple conclusions.

It also suggests a shortcoming of the current model, and the need for a regularizar to

make the results more predictable.

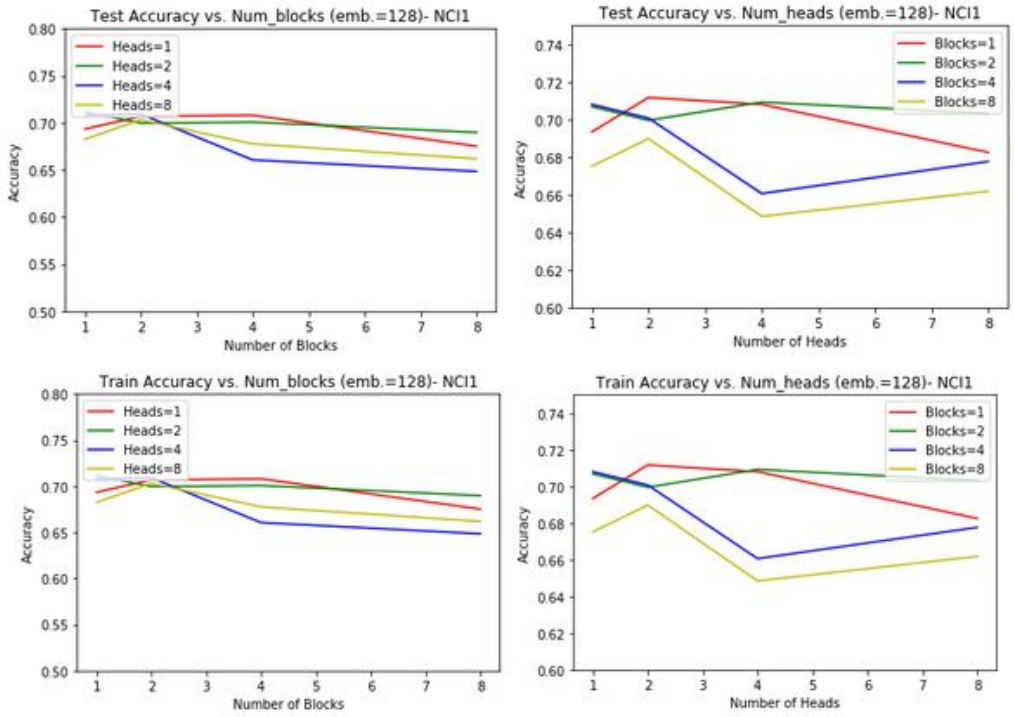


Figure 6.2: Plots for performance vs. Hyper-parameter settings for NCI1 dataset. As network becomes deeper, its performance becomes inversely proportional to number of heads and vice versa

CHAPTER 7

Conclusion

Although there is a lot of potential in deeper models, the biggest problem with deep models is over-fitting. A strong regularizer needs to be set to ensure proper training of models. This is also evident with the accuracy vs. model parameter plots. Although a strong inductive basis behind a network itself acts as a reglarizer, network performance is key to validating these concepts and sadly we were not able to achieve it with this network.

CHAPTER 8

References

1. Semi-Supervised Classification with Graph Convolutional Networks
2. Inductive Representation Learning on Large Graphs
3. Graph Attention Networks
4. Graph Isomorphism Network
5. Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks
6. Neural Message Passing for Quantum Chemistry
7. Hierarchical Graph Representation Learning with Differentiable Pooling
8. Graph U-Nets
9. Self-Attention Graph Pooling
10. Understanding Attention and Generalization in GNN
11. Citation Networks
12. TU Dortmund Graph Repository
13. Graph Neural Networks With Fast Spectral Localization
14. Gated Graph Sequence Neural Networks
15. DeepGCNs: Can GCNs Go as Deep as CNNs?
16. GResNet: Graph Residual Network for Reviving Deep GNNs from Suspended Animation
17. Relational inductive biases, deep learning, and graph networks
18. Attention is all you need
19. Deep Residual Learning for Image Recognition
20. DeepWalk: Online Learning of Social Representations