Chapter 3

problem formulation

This chapter gives a formal definition of the problem of interest and discusses some key aspects of the problem.

3.1 Problem Definition

Input. We are given two attributed graphs $G = (V, E, A)$ and $G' = (V', E', A')$, where V is the set of vertices, E represents the set of edges, and A is the set of attributes describing the vertices and edges. Denote $|V| = N$ and $|V'| = N'$. For any connected subgraph g of G with $|g| = n$ our goal is to find the top-k distinct connected subgraphs $\{g'_i \mid i \in [1,k], |g'_i| = n\}$ of G' such that any other subgraph g'_k of G' satisifes $similarity(g, g'_i) \leq similarity(g, g'_k)$. In this work, we consider only the connectivity and omit edge attributes. It is a potential future research orientation.

Graph Similarity Measure. Most existing graph similarity measures are computed based on node similarities. Mathematical approaches[][] handcraft a distance measure between nodes, then compute a $N_1 * N_2$ distance matrix M where M_{ij} is the similarity between node i in G and node j in G'. They then use approximation algorithms to find a matching between nodes that minimizes the total distance measure. Besides high space and time complexity, the main problem with this is the tedious and unscalable distance measure process.
Improving upon this, State-of-the-art machine learning models such as GMNwVGG[] adopts a similar approach by estimating a close variant of M and then perform matching matching. This M is defined as a $NN' * NN'$ matrix, where $M_{ia,jb}$ denotes how well each pair of nodes $(i, j) \in E$ matches $(a, b) \in E'$, other pairs which do not form edges are set to 0. Hence, The diagonal entries represent the matching between nodes while the others represent matching between edges. These approaches can achieve outstanding result in matching V and V'. However, in our problem, we aim to find a connected subset of V', The connectivity constraint may not be met by the above method. Even though unconnected nodes have values of 0 in M, if node features are more significant, the structure may be neglected in the matching result.
Instead of learning a similarity measure between individual nodes, this work(add model name) attempts to treat the graphs as a whole and learn the basis on which

pairs of graphs are matched given the training data. In this way, depending on what the user wants to match: feature, structure or both, the model can give optimal results for each case.

More sections.

Chapter 4

Preliminaries

This section introduces the related neural network models for this project. For each model, we analyze their working principle, application areas as well as limitations.

4.1 Graph Neural Network

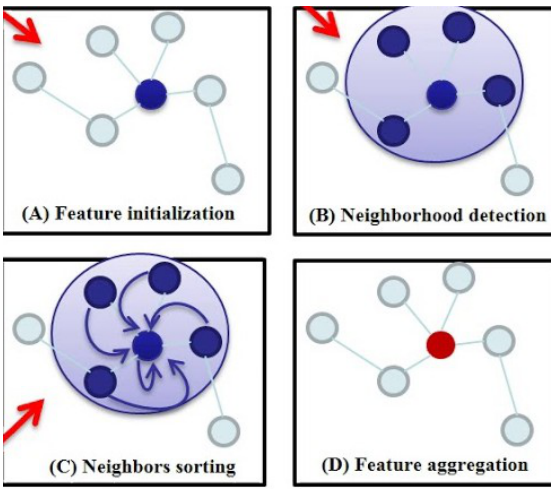

Figure 4.1: An illustration of 1-degree (hop) graph neural network[]. The model works based on the principle of message passing.

Graph Neural Network(GNN), as suggested by its name, is a machine learning model specifically dedicated to processing graphs. It uses propagation to learn about neighbourhood information (maybe move to background - One of the main contributors Scarselli et al.[] proposed using propagation to learn node representations in graphs. It has been rapidly gaining popularity among researchers after modern deep learning methods were incorporated into the model[][].) Fig 3.1 shows the standard 4-stage process for an 1-hop GNN conceptually. 1-hop means that for each node, only information about its immediate neighbours are propagated. The model takes in a graph and its adjacency matrix as input, it outputs vector embedding of every node. In stage A, we initialize the node and edge attributes of the graph. We then find the

neighbours for each node. Stage C shows message propagation to the central node from its neighbours, each message contains information about the neighbour and the edge. The last stage shows that the central node aggregates information it just collected and combine with its own information. For a n-hop GNN, the last two stages are repeated n times.
More formally, given a graph $G = (V, E)$ and its adjacency matrix A, where $i \in V$ is represented as a feature vector n_i and $(i, j) \in E$ is represented as an edge vector x_{ij} , there are two main components defined as the followed.

4.1.1 Encoder

The purpose of encoder is to map input nodes and edges features into vectors.

$$h_i^0 = f_1(n_i) \quad \forall i \in V \tag{4.1}$$

$$e_{ij} = f_2(x_{ij}) \quad \forall (i, j) \in E \tag{4.2}$$

Here, f_1 and f_2 are custom encoding functions. Depending on the context, it may be as simple as direct mapping to more widely used ones such as multilayer perceptron(MLP)[]. h_i^k is the node vector after the k-th iteration of propagation. For 1-hop GNN, k can only take values 0 and 1. In contrast, wedo not record multiple edge vectors because they do not change across propagation iterations.

4.1.2 Propagation Layer

A propagation layer performs one round of message passing and map the set $\{h_i^k \mid i \in V\}$ to $\{h_i^{k+1} \mid i \in V\}$.

$$h_i^{k+1} = f_a(h_i^k, f_m(h_i^k, h_j^k, eij)) \quad \forall i \in V, (i, j) \in E \tag{4.3}$$

f_m is a function which takes in the current vector representation of a pair of nodes and the edge vector then computes the message passed from i to j in the current iteration. It is often a MLP which takes in a concatenation of the three inputs. f_a aggregates all the messages received by i and generate a new vector representation for the next iteration. It can be either a MLP or memory-incorporated architectures like RNN[]. One has to note that the message passing is simultaneous for all nodes in one iteration.

For a n hop GNN, propagation is performed n times. The final set $\{h_i^n\}$ is the computed node embedding. Often one also wants to compute the graph level embedding, this process can be done using aggregation of node embeddings, MLP or more complex designs such as [].

4.1.3 Analysis of GNN

GNN adopts a propagation strategy to capture structural information of the input graph, its affinity towards the data structure makes it a natural choice for graph learning. One problem with GNN is that the relationship between far away nodes are not captured. Consider two nodes that are three nodes apart, using 1-hop GNN does not study the relation between them at all. Theoretically, increase the number of hops can always solve this problem. However, with each additional hop, the time complexity of the algorithm increases exponentially because the of increase in the neighbourhood size. This problem can be significant in scenarios like when immediate neighbours are induced from important nodes elsewhere.

4.2 Graph Matching Network

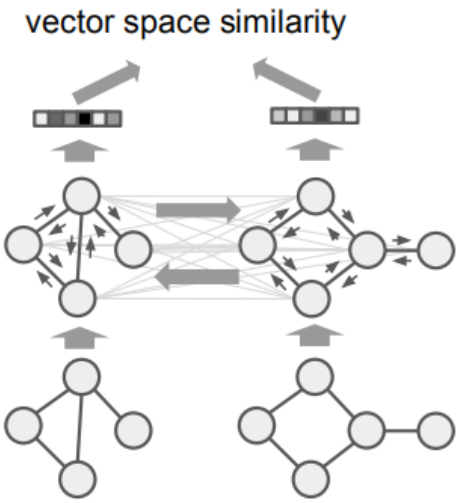

Figure 4.2: Graph Meural network[]. Use attention mechanism for cross-graph message passing.

Graph Matching Network(GMN) is an recent architecture developed by Li et al.[] specifically for the purpose of measuring the similarity between two graphs. It takes in a pair of graphs $G = (V, E)$ and $G' = (V', E')$, and outputs the similarity measure between them. The model trains two graphs jointly. From Fig 4.2, we can observe that for each input graph, a GNN is performed on it. In addition, there are arrows pointing across graphs. These arrows represent the correspondence learned between matching nodes across graphs. The encoder layer is the same as stated before and the passing of cross-graph messages is added to the propagation layer given by formula 4.4.

$$\mu_{j \to i} = f_{match}(h_i^k, h_j^k), \forall i \in V, j \in V' or\ i \in V', j \in V \tag{4.4}$$

$\mu_{j\to i}$ represents the cross-graph message from node j to$i$, where j and i belong to different graphs. f_{match} is a function which computes the information. Apparently, normal functions like MLP does not work well here because not all pairs of nodes in both graphs are an equal match to each other. Li et al. proposes using an attention mechanism module.

$$a_{j\to i} = \frac{exp(s_h(h_i^k, h_j^k))}{\sum_{j'} exp(s_h(h_i^k, h_{j'}^k))}\mu_{j\to i} = a_{j\to i}(h_i^k - h_j^k)\sum_j \mu_{j\to i} = \sum_j a_{j\to i}(h_i^k - h_j^k) = h_i^k - \sum_j h_j^k \tag{4.5}$$

s_h is a similarity measure, GMN uses Euclidean distance. $a_{j\to i}$ is the attention weight measuring the strength of correspondence between j and i. It is computed using a softmax-based approach. The reason is that for i and j, the more similar their vector representation, the more likely they are matching ones. Therefore, more attention is paid to the difference between i and its hypothetical match j. This difference, timed by the attention weight is the message received by i from j. The extreme case where $G = G'$ will give a peaked attention value at the matching node and the total message passed across graphs will be 0, further validating the approach.The model then proceeds to compute graph-level embeddings of G and G' and computes the similarity using Euclidean distance.

4.2.1 Analysis of GMN

According to [], GMN has attained outstanding performance on the task it is designed to do: measuring similarity between graphs at the expense of higher complexity associated with computing attention for each pair of nodes across the graphs. In my opinion, in addition to resource consumption, one significant drawback with GMN and other existing GNN-based graph similarity measures is they are like a blackbox which do not account for what contributes to the matching score. Questions like whether it is the similarity in structure or the features, which features are contributing the most are left unanswered.

4.3 Long-Short Term Memory

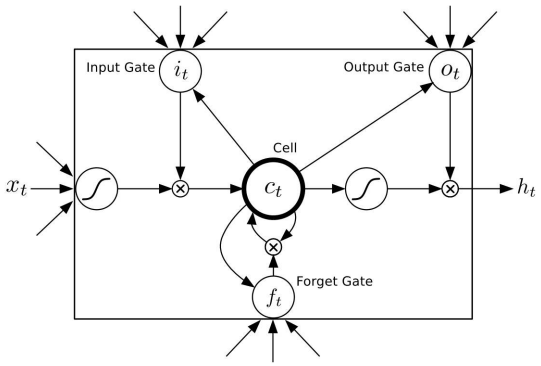

Figure 4.3: A LSTM unit[]

Add LSTM functions and explain the graph.

4.4 Wasserstein embedding

Add explanation

4.5 GMNwVGG

Add explanation, omit tedious back propagation

Chapter 5

Design and Implementation

Revisiting the problem definition, we are given two attributed graphs $G = (V, E, A)$ and $G' = (V', E', A')$, where V is the set of vertices, E represents the set of edges, and A is the set of attributes. For any connected subgraph g of G with $|g| = n$ our goal is to find the top-k distinct connected subgraphs $\{g'_i \mid i \in [1,k], |g'_i| = n\}$ of G' such that any other subgraph g'_k of G' satisifies $similarity(g, g'_i) \leq similarity(g, g'_k)$. Since we are matching subgraphs, it is necessary to have pairs of subgraphs as the training data. The process of obtaining such pairs will be discussed in the next chapter. We denote the training pairs as $\{(g_k, g'_k)\}$ with $|g_k| = n$ and let the dimension of the feature encoding of each node to be m, therefore, each graph can be represented as a $n * m$ matrix.
This section discusses the design of (model name)model to solve the problem and the reason for this design.

5.1 Model Overview

(model name) has two main components. It takes in G, G' the target graphs as well as the matching pairs $\{(g_k, g'_k)\}$. When testing with a subgraph $g \in G$, the model output is the node representations of nodes in g which can then be use to find g' in G'. The first component focuses on learning the graph features using Graph Neural Network(GNN). The second part of the model aims to find the matching subgraphs through learning. Inspired by approaches in natural language processing(NLP), it treats graphs as sequences of nodes and processes them with a Long Short-Term Memory(LSTM) to study the correlation between the nodes.

5.1.1 Feature Learning

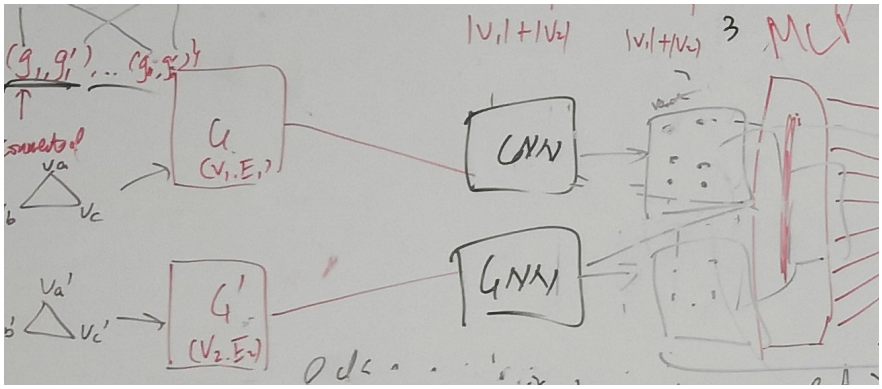

Figure 5.1: to be changed into a better one with clear label

Figure 3.1 shows the structure of the embedding section. In general, it consists of two steps: using GNN to learn the node features separately, then treat them jointly with a MLP layer.

5.1.1.1 Feature Learning using GNN

We denote the two GNNs as f_1 and f_2 . For the encoder of GNN, we use a single layer multilayer perceptron. The output of the two GNNs for a training pair (g_k, g'_k) are hence

$$h_k = f_1(g_k, A) = f^n_{prop}(MLP1(n_i)\ \forall i \in V_{g_k}, MLP2(e_{ij})\ \forall (i,j) \in E_{g_k},) h'_k = f_2(g'_k, A') = f^n_{prop}(MLP1'(n'_i)\ \forall i' \tag{5.1}$$

where n is the number of propagation iterations or n-hop, f_{prop} is the propagation layer as defined in section 4.1.2. The input to the propagation are the initial node vectors and edge vector generated from the MLP layers. All the MLP layers are separate. Assume the length of node vector is l, the outputss h_k and h'_k are $n * l$ matrices where each line denotes the embedding of the corresponding node. The reason for using separate GNNs is because they belong to separate graphs, combining them would not make a difference as there are no message passing across graphs. An alternative(potentially better) option is using GMN for the same purpose.

5.1.1.2 Combine the Embeddings

Next, we join the embeddings because looking atg_k and g'_k separately does not help understand how they are matched. This process is done by simply concatenating the inputs along the first dimension and then pass through a MLP layer.

$$out = MLP([h_k|h'k]) \tag{5.2}$$

The concatenated input has shape $2n * l$. The shape is preserved in the output. This is the final output of the feature learning component.

5.1.1.3 (Wasserstein Embedding Layer)

5.1.1.4 Embedding Loss

In the output $2n * l$ matrix, the first n rows are the embedding E_{g_k} of g_k and the rest are $E_{g_{k'}}$ for g'_k .We calculate the embedding loss after this stage. The loss $L_e$is defined jointly for the training pair.

$$L_e = ||g_k - g'_k||_2 \tag{5.3}$$

Here, we use the Euclidean distance to measure similarity between the embeddings.

5.1.2 Structure Matching

Though GNN is capable of capturing some degree of structural information, its deficiencies including neglecting "far away" nodes and the lack of lower level matching details means that it is not enough for our problem.
Therefore, the second part of (the model) aims to capture more structure information of graphs. To solve the problem that GNN does not capture node relations far away, we are inspired by text processing in natural language processing(NLP). In text processing, one of the most widely used approaches is treating the text as a sequence of tokens. The high correlation of words and phrases in everyday language leads to the fact that tokens are usually dependent on the token before. Sometimes, these dependencies are long term. If graphs can be treated as sequential data similarly, the rich resort of NLP methods may be used for this problem as well. Coincidentally, nodes in a connected graph are pairwise related because there is always a path between any pair. The length of paths could to some extent denote long term and short term dependencies. We use the popular LSTM model to learn about the correlation between nodes.

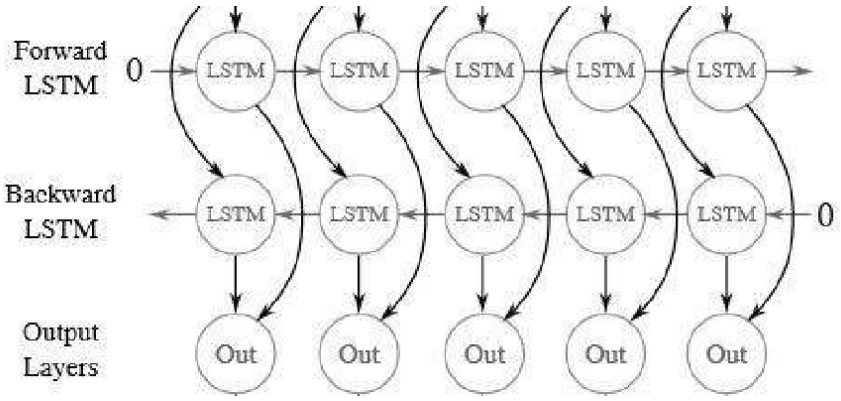

Figure 5.2: 1 layer Bi-directional LSTM used in the model.[5] The input is one subgraph in each pair.

Fig 4.2 depicts the structure of the 1 layer bi-directional LSTM used in the model. The input to this part is one output graph form the previous stage E_{g_k} of shape $n * l$. Therefore, there are n timesteps in the LSTM model. The sequence of nodes in g_k may not be ideal since there is no guarantee that nodes before do not have dependencies on the nodes after. Therefore, I choose bi-LSTM instead of the normal one because feeding the input in both directions can theoretically capture dependencies in both directions. The output size of the LSTM is set to be l so that it can be used to find matches of the node. The output O after the last timestep, a $n * l$ matrx, is taken as the output. **Add formula here**

5.1.2.1 Matching Loss

For each output O produced, we compute the matching loss as followed.

$$g'' \ = \ \{n' \mid \forall z' \in G'.\forall i' \in g_k.d(z', i) >= d(O(n'), i)\} L_e \ = \ ||g'' - g'|| \tag{5.4}$$

g'' is the subgraph found by the model. It is compute as the set of nodes which are the nearest neighbour to rows in O, in case there are duplicates, the second closest for a row is used instead.

The total loss of the model is $L_e + \lambda L_m$, where λ is a hyperparameter between 0 and 1 to be optimized through validation.

5.2 Section summary

(To be updated as I progress)

Chapter 6

Experimental Methodology

This chapter describes the implementation of the (model name) model as well as detailed experimental results on a synthetic dataset and Cora dataset. We evaluate and compare the result of the model, both quantitatively and qualitatively, with three baselines: (a mathematical algorithm)[], GMNwVGG[] and GMN[].

6.1 Datasets

6.1.1 Synthetic Dataset

The problem takes attributed graphs G and G' as inputs, but the model trains on pairs of their subgraphs.

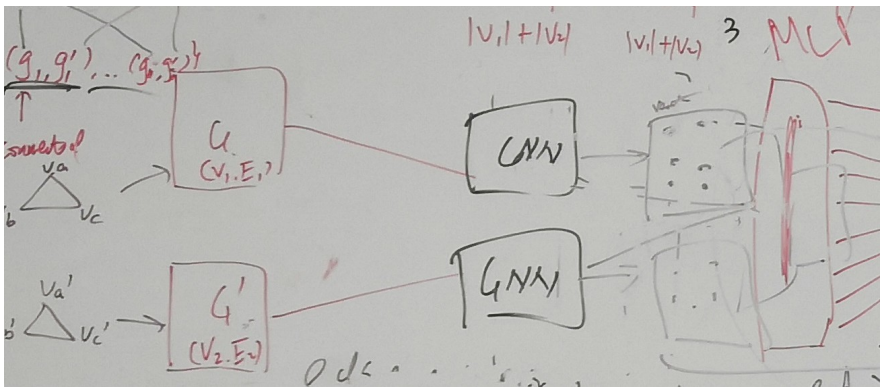

Figure 6.1: Placeholder, to be changed into psuedocode later

Fig 5.1 shows the algorithm for dataset generation. The generator takes in 7 parameters, as listed in table 5.1. **input table 5.1 here**

We start with G and G' both empty, then perform a recurrent process of two steps:
Step 1: Generate one pair of matching subgraphs (g, g')
Step 2: Aggregate g into G, g' into G'

6.1.1.1 Pair Generation

Line (a to b) in Fig 5.1 generates two random matching connected graphs g and g′. This is achieved by generating g first and then construct a matching one g′ based on matching criteria decided by w_n and w_a.

Explain how to use the two parameters here (Put into table: w_n denotes the probability that attribute values are the same for nodes with the same id. If w_n is 1, nodes with the same id in both graphs have the same attributes values and if it is 0, the node attributes in g′ are randomly selected from the domain excluding the value taken by the corresponding node in g.(example).)

(Put into table: w_s is the probability that an edge in g is kept in g′ and g′ containing an edge which does not appear in g. If it is 1, g and g′ have the exact same edges and if it is 0, both graphs do not contain the same edges unless they are necessary to keep g′ connected. (example))

6.1.1.2 Graph Aggregation

Line (c to d) performs the graph aggregation step. Each time when generating a new pair of subgraphs g and g′, an existing node is selected from G and G′ with the same node id and reused in the new pair.

For the synthetic dataset used in this experiment, each node has one attributes "color" with domain$\{"R","G","B"\}$. Edge attributes are omitted. Three sets of values for w_n and w_s are used to evaluate the performance: 0 0, 0.5 0.5 and 1 1. For each set , the dataset contains 50 pairs of matching subgraphs, each with 5 nodes.

6.1.2 Cora Dataset

Cora[] is a well-known node classification dataset. It contains 2708 machine learning papers each classified into one of seven classes. The dataset uses a dictionary of 1433 unique keywords as paper features.Hence, each paper can be viewed as a node represented by its id and binary values denoting the presence of keywords. The dataset also records the citation relation among papers, which forms the edges. We modify the original dataset to fit into our model using the algorithm in Fig 5.2.

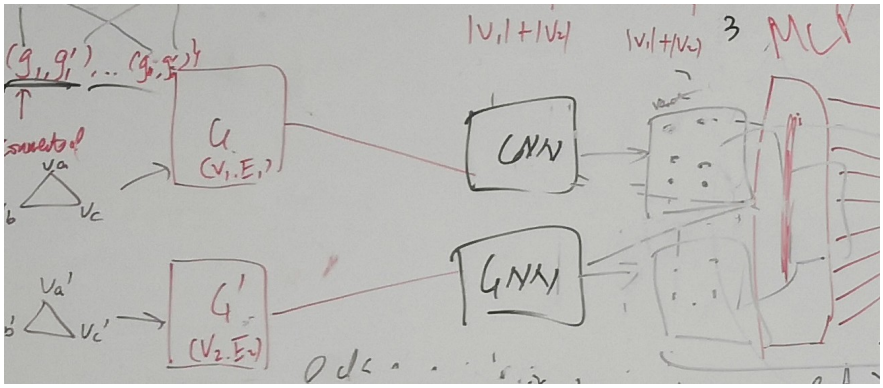

Figure 6.2: Placeholder, to be changed into psuedocode later

6.2 Experimental Setup

state the parameters chosen and software