IE683 - Topics in Learning Algorithm

Distance based Graph Learning

By

Saurabh Adhikary (203196002)
Sajith Menon(203190025)
Rachit Kumar(193190023)

Supervisor:
Prof. Nandyala Hemachandra

Table of content

Introduction

Graph allows relational knowledge about their interacting entities to be efficiently stored and accessed.Many machine learning applications seek to form predictions or discover new patterns using graph-structured data as feature information.Social networks, molecular graph structures, biological protein-protein networks can be easily modelled using graph.The main problem with graph learning is to find a way to represent encode its structure so we can easily use it for our machine learning model.In this project we have discussed up on advantages of graph learning , challenges behind implementing and representing it, learning distance of a graph and several other Embedding technique involved in graph learning.

Traditional approach to graph learning has its drawbacks,so we use representation learning to overcome its challenges.Graphs are non-Euclidean discrete data structures, and several works have been proposed to learn graph embeddings into non-Euclidean spaces rather than conventional Euclidean space. We also studied techniques in learning Distance in graph.We use Siamese Architecture for this purpose. Error tolerant or inexact graph matching algorithms are proposed to cope with deformations between graphs. The problem consists in finding the minimum transformation cost such that an isomorphism exists between the transformed graph and the second one.Graph edit distance Algorithm are widely used as error-tolerant graph matching methods. But the drawback in Ged Algorithm is its time complexity is exponential. Hence, GED is unfeasible in a real scenario without constraints in terms of graph size.Convolutional neural network has been extended to use it for solving graph problems. These new extensions are also known as Geometric Deep learning.

Notations & Assumptions

Input to our representation learning algorithm is an undirected graph G = (V , E) and binary adjacency matrix ‘A’ for internodal weights.

Node attributes X $\in R^{m \times |v|}$

where,

m - no of data samples

v - no of features

ENC : V $\rightarrow R^{d}$

It is Encoder function which maps nodes to vector embeddings zi $\in R^{d}$

DEC : $R^{d} \times R^{d} \rightarrow R^{+}$

It maps pairs of nodes embedding to a real valued similarity measure,which provides us with the similarity of the two nodes in the original graph.

$\mathbf{S_G}$: V $\times$ V $\rightarrow R^{+}$

It gives us a measure of similarity between nodes in G.

Distance Learning using Graph

We know that Graphs are symbolic representations.Unlike vectorial representations such as SIFT or HOG they are more complex.Graph have underlying non euclidean distances.But our Current Machine learning model works better with Euclidean distances so we use methods such as graph neural network so our information not lost while transforming it into Euclidean input. When dealing with graphs, an important property is the ability to compare two graphs in terms of a similarity or distance.So it is very much important to learn distances in graphs.Ged Siamese Mpnn Geometric Deep learning are some of the techniques we use to find out Distances in graph.We will discuss about this technique in more detail.

Siamese Architecture

Distance learning for graphs can be found out with a siamese architecture.This Architecture got inspired by earlier work in distance learning for images with siamese neural networks.In Siamese architecture it uses the same model and weights to learn a representation where distances can be computed.These networks are used to find the similarities by comparing their feature vectors.A twin Message passing neural network(MPNN) with shared weights is applied to the input graphs. Message passing neural networks(MPNNs) have the ability to update the hidden state of a particular node ‘v’ with the information of its neighbourhood sent through a particular edge ‘e’. Through the various time steps, the system is gathering structural information of the local context of the node. Thus, the Message passing neural network(MPNN) approach is learning an enriched representation of the original graph.

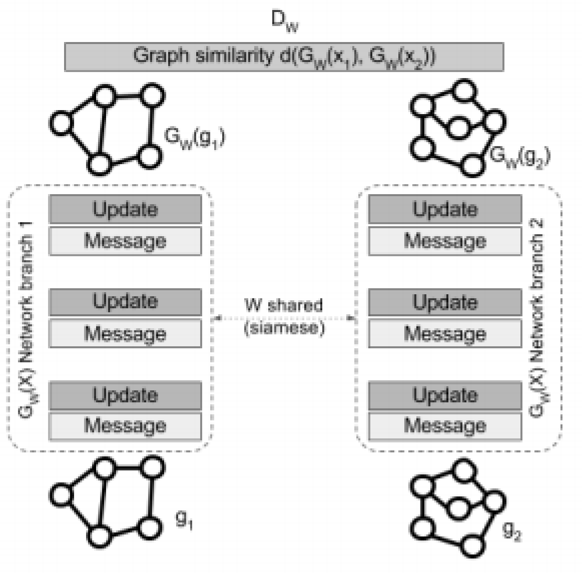

Figure 1 - Architecture for the proposed siamese MPNN model

Readout function given by,

$$R = \sum_{v \in V} \sigma(i(h_v^{(T)}, h_v^0)) \odot (j(h_v^T)),$$

where i and j are neural networks and σ is the Sigmoid activation function,is used to map our graph ’g’ into vector spaces has mainly two drawbacks:First we see that there is no node correspondence between the graphs; and second one is that individual properties for nodes and edges are not taken into account. To avoid these drawbacks, we altogether discard the readout phase and directly compute a distance between enriched graph representations provided by the message passing phase. Therefore, we use Graph Edit Distance(GED) idea to obtain a similarity metric between graphs.The message function in Mpnn is formulated as $M(h_v, h_w, e_{vw}) = A(e_{vw})\ h_w$, where A_{evw} is a learned matrix for each possible edge label. To overcome the constraints put by this function Gilmer et al changed this message function as $M(h_v, h_w, e_{vw}) = A(e_{vw})\ h_w$ to allow vectorial data as edge attributes, where $A(e_{vw})$ is a neural network which maps the edge vector to a matrix. The update function that we are using here is defined by $U(h_v, m_v) = GRU(h_v, m_v)$, where GRU is the Gated Recurrent Unit.

Graph Edit Distance:

Graph edit distance (GED) calculates the similarity of two graphs in terms of edit operations. The distance Computation is done by what we call a string edit distance formulation. Our main idea is to compute the minimum cost transformation from the source graph to the target one in terms of a sequence of edit operations.Common edit operations are node and edge insertion, deletion and substitution. Each edit operation has an associated that is added to the final edit distance at every step. Among all the combinations of edit sequences, our distance is formulated in terms of the minimum cost edit sequence. GED algorithm finds an optimal edit sequence, but the main drawback with GED is that computational complexity of it is exponential in the number of nodes of the involved graphs.

$$\mathrm{d}(G_1, G_2) = \min_{(e_1,\ldots,e_k)\in \Upsilon(G_1,G_2)} \sum_{i=1}^{k} \mathrm{c}(e_i)$$

where Y(G1, G2) denotes the set of edit paths transforming G1 into G2, and c is that the cost function of the edit path ei . Since this algorithm is computationally very expensive we use many approximation techniques such as Bipartite graph matching and Hausdorff Edit Distance to compute approximation of GED.

Hausdorff distance

Hausdorff is simple but effective metric based technique.The Hausdorff distance of two sets A and B on a metric space, with the metric d(a,b) where a ∈ A and b ∈ B is defined as

$$\mathrm{H}(A,B) = max\left(\sup_{a\in A}\inf_{b\in B} d(a,b), \sup_{b\in B}\inf_{a\in A} d(a,b)\right)$$

$$\mathrm{H}(A,B) = max\left(\max_{a\in A}\inf_{b\in B} d(a,b), \max_{b\in B}\inf_{a\in A} d(a,b)\right)$$

Hausdorff distance is very sensitive to outliers. Replacing the maximum operator with the sum, forces the distance to take into account all nearest neighbour distances and becomes more robust to noise than the original one. Thus, we can define the distance as:

$$\hat{\mathrm{H}}(A,B) = \sum_{a\in A} \inf_{b\in B} d(a,b) + \sum_{b\in B} \inf_{a\in A} d(a,b)$$

Therefore, we define the distance between two graphs $g_1 = (V_1, E_1, \mu_1, \nu_1)$ and $g_2 = (V_2, E_2, \mu_2, \nu_2)$ as:

$$d(g_1, g_2) = \frac{\hat{\mathrm{H}}(V_1, V_2)}{|V_1| + |V_2|}$$

The distance d can be seen as a specific case of HED where all nodes must be substituted and there are no insertions or deletions. Moreover, the edges are not taken into account because the local structure exploited by HED has been embedded during the message passing phase.

The proposed approach is trained with a supervised manner knowing whether or not a pair of graphs belong to an equivalent class. All the models were trained using the Stochastic Gradient Descent optimiser with Momentum and weight decay i.e. L2 regularization. The proposed objective function that is minimised is a Contrastive loss function based on the proposed distance, and it is defined as:

$$l(D_W) = Y\frac{1}{2}(D_W)^2 + (1-Y)\frac{1}{2}\{\max(0, m - D_W)\}^2$$

where $D_w = d(G_w\ (X1),\ G_w\ (X2))$ is a distance defined between the outputs of the message phase of our model G_w with shared weights W; Y $\in$ {0, 1} is a label indicating positive or negative pairs of graphs, i.e. they belong to the same class; and 'm' is a margin. In this work, 'm' is set to 1.0.

Graph-based Keyword Spotting

Keyword Spotting (KWS) does the task of retrieving instances of a given keyword in a document without explicitly transcribing it.Now a days Graph representation for KWS is gaining immense popularity. Wang et al represented handwritten words using characteristic skeleton points starting/ending, high-curved and branch points, labelled with shape context features.Convexity based representation is also proposed nodes are been represented by convexities labelled with shape descriptors.They known to have this problem of segmentation based keyword spotting. Recently Riba et al has proposed a wonderful idea of segmentation free approach.it is based on graph indexation.Nodes are normally represented with image coordinates.

Challenges and Solutions in Graph Learning

Challenges

Graph Learning is more complex than vectorial representation.It is difficult to represent, or encode, graph structure so that it can be easily exploited by machine learning models.it is an uphill task in automatically learning to encode graph structure into low-dimensional embeddings.it is not an easy job to use techniques based on deep learning and nonlinear dimensionality reduction to compare two graphs in terms of finding similarity or distance between them incorporate information about graph structure to Machine Learning models.

Solutions in Graph Learning

Traditional Approach

Traditionally Graph based learning problems are to extract structural information from graphs ML uses:

1. Summary graph Statistics like degree or clustering coefficient
2. Kernel functions
3. carefully engineered features.

Drawback :

1. These Hand engineered features are inflexible
2. Cannot adapt during the learning process
3. Features can turn out to be time consuming and computationally expensive
4. Solution to Biggest challenges in Graph Learning

Representation Learning

These Approaches learn a mapping that encodes Structural information about graph ideas to learn a mapping that embeds nodes or entire subgraphs as points in low dimensional vector space.

Our goal is to optimize this mapping. Representation learning approaches treat this mapping problem as a machine learning task itself, employing a data-driven approach to find out embeddings that encode graph structure.

Representation learning can be learned using Following Approaches

1. Encoder-decoder perspective
2. Shallow embedding approaches
    a. Factorization-based
        i. Laplacian eigenmaps
        ii. Inner-product methods
3. Random walk approaches
4. DeepWalk and node2vec
5. Large-scale information network embeddings (LINE)
6. HARP: Extending random-walk embeddings via graph pre-processing

Encoder-decoder

Encoder, maps each node to a low-dimensional vector, or embedding.

Decoder, which decodes structural information about the graph from the learned embeddings (majority of works use a basic pairwise decoder)

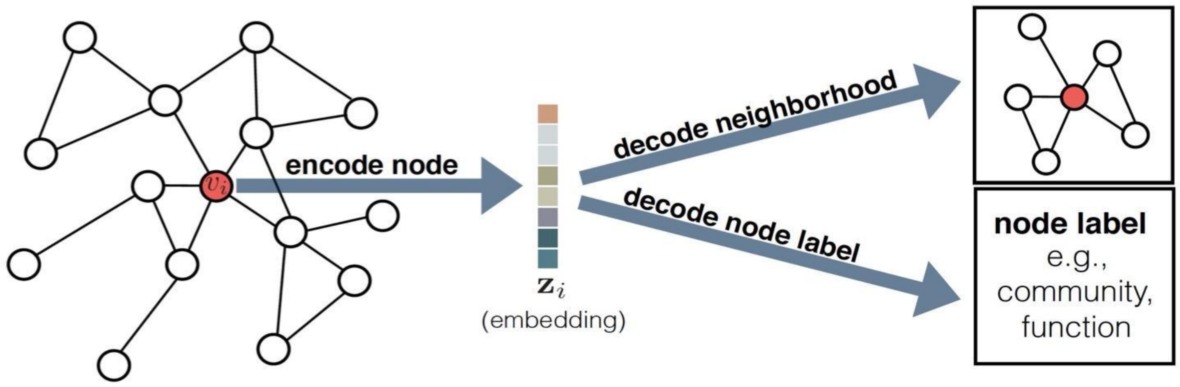

Figure - 2 Overview of Encoder Decoder Approach

ENC : V → R^d DEC : $R^d \times R^d \rightarrow R^+$

When we start applying our pairwise decoder to a pair of embeddings (zi ,zj) we get a reconstruction of the similarity between vi and vj in the original graph, and now our goal is to optimize the encoder and decoder mappings to reduce the error, or loss, during this reconstruction so that:

$$\mathrm{DEC}(\mathrm{ENC}(v_i), \mathrm{ENC}(v_j)) = \mathrm{DEC}(z_i , z_j) \approx S_G(v_i , v_j)$$

where sG is our user-defined, graph-based similarity measure between nodes, defined over the given graph G.we are trying to optimize our encoder-decoder model so that we can decode pairwise node similarities in the original graph $S_G(v_i , v_j)$ from the low-dimensional node embeddings z_i and z_j .

In real life scenario, most approaches realize the reconstruction objective by minimizing an empirical loss L over a set of training node pairs D given by :

$$\mathcal{L} = \sum_{(v_i,v_j)\in\mathcal{D}} \ell\left(\mathrm{DEC}(\mathbf{z}_i, \mathbf{z}_j), s_{\mathcal{G}}(v_i, v_j)\right),$$

where `l : R × R → R is a user-specified loss function, which measures the discrepancy between the decoded (i.e., estimated) similarity values $\mathrm{DEC}(z_i , z_j)$ and the true values $S_G(v_i , v_j)$.After Optimizing our encoder decoder system we use this trained encoder and generate embeddings for nodes which we later use in machine learning task as our feature input.we could use our learned embeddings in a logistic regression classifier to predict which community a particular node belongs.

Shallow embeddings

Most of the node embedding we find rely on shallow embedding.In shallow embedding Approach encoder function maps to vector space is simply an “embedding lookup”

ENC(vi) = Zvi ,

where Z ∈ R d×|V| is our matrix containing the embedding vectors for all nodes and vi ∈ IV is our one-hot indicator vector indicating the column of Z corresponding to its node vi . The set of trainable parameters for shallow embedding methods is simply ΘENC = {Z}, i.e., the embedding matrix Z is optimized directly.Shallow embedding is used by Factorization methods and Random walks.

(a) Factorization-based : In Early days lot methods in learning representations for nodes were largely focused on matrix-factorization approaches, it is somewhat inspired from Dimensional reduction.

(i)Laplacian eigenmaps : it uses shallow embedding technique which is defined by

$$\mathrm{DEC}(\mathbf{z}_i, \mathbf{z}_j) = \|\mathbf{z}_i - \mathbf{z}_j\|_2^2$$

And loss function is given according to similarity of graph

$$\mathcal{L} = \sum_{(v_i, v_j) \in \mathcal{D}} \mathrm{DEC}(\mathbf{z}_i, \mathbf{z}_j) \cdot s_{\mathcal{G}}(v_i, v_j).$$

(i)Inner-product methods

In Laplacian eigenmaps technique, there are huge number of recent embedding methodologies that supported a pairwise, inner-product decoder

DEC(zi , zj) = z > i zj ,

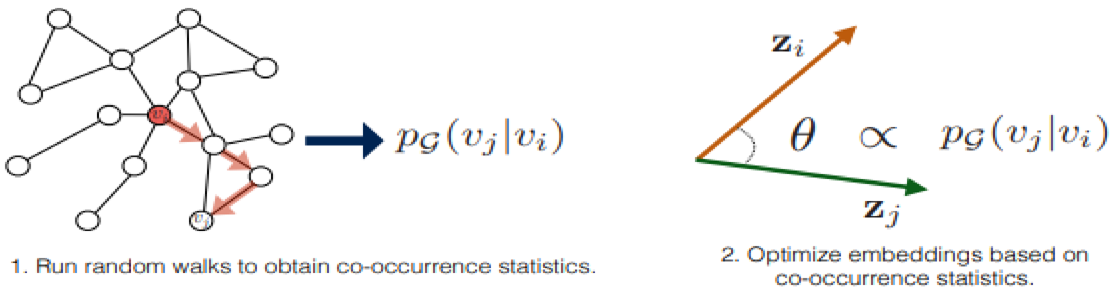
$p_{\mathcal{G}}(v_j|v_i)$
1. Run random walks to obtain co-occurrence statistics.
$\mathbf{z}_i$
$\theta \propto p_{\mathcal{G}}(v_j|v_i)$
$\mathbf{z}_j$
2. Optimize embeddings based on co-occurrence statistics.

Figure 3 Inner Product Method

Strength of relationship between two nodes is given by their dot product of embeddings.Graph factorization algorithm Hope Grarep are some of the methods which uses inner product.it uses MSE as a loss function

$$\mathcal{L} = \sum_{(v_i,v_j)\in\mathcal{D}} \|\mathrm{DEC}(\mathbf{z}_i,\mathbf{z}_j) - s_{\mathcal{G}}(v_i,v_j)\|_2^2,$$

Random walk :

The random-walk based methods sample a large number of fixed-length random walks starting from each node, vi. The embedding vectors are then optimized so that the dot-product, or angle, between two embeddings, zi and zj, is (roughly) proportional to the probability of visiting vj on a fixed-length random walk starting from v

D is generated by sampling random walks starting from each node (i.e., where N pairs for each node vi are sampled from the distribution (vi,vj) ← pG,T(vj|vj))

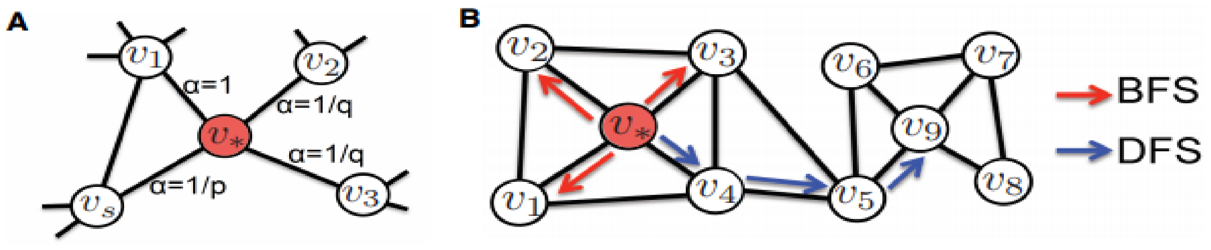

Figure 4 -Illustration of node2vec biasing random walk

DeepWalk and Node2vec

DeepWalk and node2vec also use shallow embedding and in particular it uses a decoder based on the inner product.s. The basic idea behind Deepwalk and Node2vec approaches is to learn embeddings so that:

$$\mathrm{DEC}(\mathbf{z}_i,\mathbf{z}_j) \triangleq \frac{e^{\mathbf{z}_i^\top \mathbf{z}_j}}{\sum_{v_k\in\mathcal{V}} e^{\mathbf{z}_i^\top \mathbf{z}_k}} \approx p_{\mathcal{G},T}(v_j|v_i),$$

And it also attempt to minimize the cross-entropy loss:

$$\mathcal{L} = \sum_{(v_i,v_j)\in\mathcal{D}} -\log(\mathrm{DEC}(\mathbf{z}_i, \mathbf{z}_j)),$$

Deepwalk employs a “hierarchical softmax” technique to compute the normalizing factor, using a binary-tree structure to accelerate the computation.Node2vec approximates the cross entropy loss using “negative sampling”: instead of normalizing over the full vertex set, node2vec approximates the normalizing factor using a set of random “negative samples”.node2vec introduces two random walk hyperparameters, p and q, that bias the random walk . The hyperparameter p controls the likelihood of the walk immediately revisiting a node, while q controls the likelihood of the walk revisiting a node’s one-hop neighborhood

By introducing these hyperparameters, node2vec is able to smoothly interpolate between walks that are more close to breadth-first or depth-first search.

Large-scale information network embeddings (LINE)

LINE combines two encoder-decoder objectives that optimize “first-order” and “second-order” node similarity.

The first-order objective uses a decoder that is based on the sigmoid and an adjacency-based similarity measure

The second-order encoder-decoder objective is similar but considers two-hop adjacency neighborhoods and uses an encoder like a softmax fn

Shallow embedding - drawbacks

1. No parameters are shared between nodes within the encoder (i.e., the encoder is just an embedding lookup based on arbitrary node ids). This can be statistically inefficient, since parameter sharing can act as a strong sort of regularization, and it’s also computationally inefficient, since it means the amount of parameters in shallow embedding methods necessarily grows as O(|V|).
2. Shallow embedding also fails to leverage node attributes during encoding. In many large graphs nodes have attribute information (e.g., user profiles on a social network) that is often highly informative with respect to the node’s position and role in the graph.
3. Shallow embedding methods are inherently transductive , i.e., they can only generate embeddings for nodes that were present during the training phase, and they cannot generate embeddings for previously unseen nodes unless additional rounds of optimization are performed to optimize the embeddings for these nodes. This is highly problematic for evolving graphs, massive graphs that can't be fully stored in memory, or domains that need generalizing to new graphs after training.

Till now we have seen node Embedding Technique where our goal was to encode nodes as low-dimensional vectors that summarize their graph position and the structure of their local graph neighborhood.

Now Let us review some methods of subgraph embedding.Here,our goal is to learn a continuous vector representation.unlike the node embedding setting, most subgraph embedding approaches are fully Supervised where the goal is to predict a label associated with a particular subgraph.We are Assuming these embeddings are being fed through a cross-entropy loss function

Hyperbolic Embedding

Hyperbolic embedding uses gradient based optimization technique.this approach learns via poincare distance function given by:

$$\begin{aligned} d_2(Z_i, Z_j) &= d_{\text{Poincaré}}(Z_i, Z_j) \\ &= \operatorname{arcosh}\left(1 + 2\frac{\|Z_i - Z_j\|_2^2}{\left(1 - \|Z_i\|_2^2\right)\left(1 - \|Z_j\|_2^2\right)}\right). \end{aligned}$$

Embeddings are then learned by minimizing distances between connected nodes while maximizing distance between disconnect nodes.

$$d_1(W, \widehat{W}) = -\sum_{ij} W_{ij} \log \frac{e^{-W_{ij}}}{\sum_{k|W_{ik}=0} e^{-\bar{W}_{ik}}} = -\sum_{ij} W_{ij} \log \text{Softmax}_{k|W_{ik}=0}\left(-\widehat{W}_{ij}\right)$$

Denominator of Hyperbolic embeddings are approximated using negative samplings.
They need to be optimized using Riemannian optimization techniques to ensure that they remain on the manifold.

Neighborhood autoencoder methods

To generate an embedding for a node, v_i, the neighborhood autoencoder approaches first extract a high- dimensional neighborhood vector si ∈ R|v|, which summarizes v_i's similarity to all other nodes in the graph. The s_i vector is then fed through a deep autoencoder to scale back its dimensionality, producing the low-dimensional zi embedding.

Unlike shallow embedding methods, this method directly incorporates graph structure into the encoder algorithm using deep neural networks.

In this Approach each node, v_i , is associated with a neighborhood vector, s_i .

The s_i vector contains v_i's similarity with all other nodes in the graph and functions as a high-dimensional vector representation of v_i's neighborhood.

Graph neural networks(GNN)

GNN Model was the first attempt to use deep learning techniques on graph structured data.

Instead of aggregating information from neighbors, the intuition behind GNNs is that graphs can be viewed as specifying scaffolding for a "message passing" algorithm between nodes.

It uses frameworks such as message passing networks.In First Formulation of GNN model Gori et al n views the supervised graph embedding problem as an information diffusion mechanism, where nodes send information to their neighbors until some stable equilibrium state is reached.

Given a Random initialized node embedded Z_0 following recursion applied till its convergence

$$Z^{t+1} = \text{ENC}\,(X, W, Z^t; \Theta^E)$$

After convergence (t = T), the node embeddings Z^T are then used to predict our final output such as node or graph labels given by:

$$\hat{y}^S = \text{DEC}\,(X, Z^T; \Theta^S)$$

Process Repeated Several times and GNN parameter are learned with backpropagation via Almeda-pineda Algorithm .it is guaranteed to Converge to unique solution when iteration mapping is contraction mapping.After Embeddings are converged final output embedding gets calculated given by s zvi = g(h K i), where g is an arbitrary differentiable function of the form g : R d → R d.

Li et al modified the GNN framework to use Gated Recurrent Units and backpropagation through time.

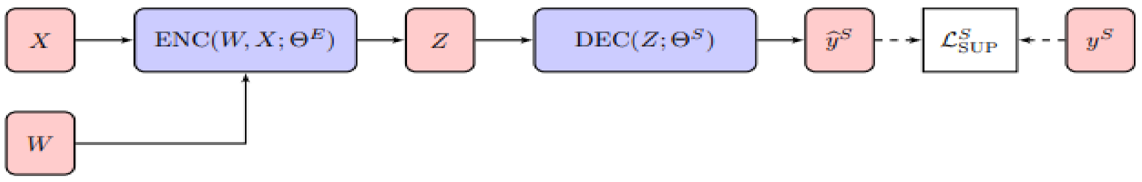
X
$\mathrm{ENC}(W, X; \Theta^E)$
Z
$\mathrm{DEC}(Z; \Theta^S)$
$\widehat{y}^S$
$\mathcal{L}^S_{\mathrm{SUP}}$
y^S
W

Figure - 5 Supervised Graph Neural network

Instead of using the graph structure to act as a regularizer, GNNs leverage the graph structure in the encoder to propagate information across neighbouring nodes and learn its structural representations. Labels are then decoded and compared to ground truth labels (e.g. via the cross-entropy loss).

Distance Based Non Euclidean Methods

a) **Lorentz Model -** Provide Better numerical stability than poincare Embeddings
b) **mixed- curvature product spaces** -provide more flexibility for other types of graphs (e.g. ring of trees).
c) **Extended Poincare Embedding** : It incorporates skip gram losses using hyperbolic inner product.
d) **Skip-gram graph embedding models** :were inspired by efficient NLP methods modeling probability distributions over words for learning word embeddings.Skip gram minimize the objective

$$\mathcal{L} = - \sum_{-K \leq i \leq K, i \neq 0} \log \mathbb{P}(w_{k-i} | w_k),$$

They exploit Random walk by producing node sequences that are similar in positional distribution.the decoder function is also an outer product and the graph regularization term is computed over random walks on the graph.they typically suffer from a high computational complexity, which makes it difficult to apply these matching algorithms in a real scenario.

Conclusion

Representational learning using graphs has a wide range of Application in Machine Learning.But graph learning still has a lot of drawback in terms of complexity in its representation.Node classification and link prediction are areas where Graph learning is been widely used.But still lot of scope is there in improving its performances.

We have studied some of its representation Embedding technique Graph distance calculation .its advantages and drawbacks which could be further explored in future to get a better graph learning model.

References

1. Riba, Pau, et al. "Learning graph distances with message passing neural networks." 2018 24th International Conference on Pattern Recognition (ICPR). IEEE, 2018.APA

2. Hamilton, W. L., Ying, R., & Leskovec, J. (2017). Representation learning on graphs: Methods and applications. arXiv preprint arXiv:1709.05584.

3. Chami, I., Abu-El-Haija, S., Perozzi, B., Ré, C., & Murphy, K. (2020). Machine learning on graphs: A model and comprehensive taxonomy. arXiv preprint arXiv:2005.03675.