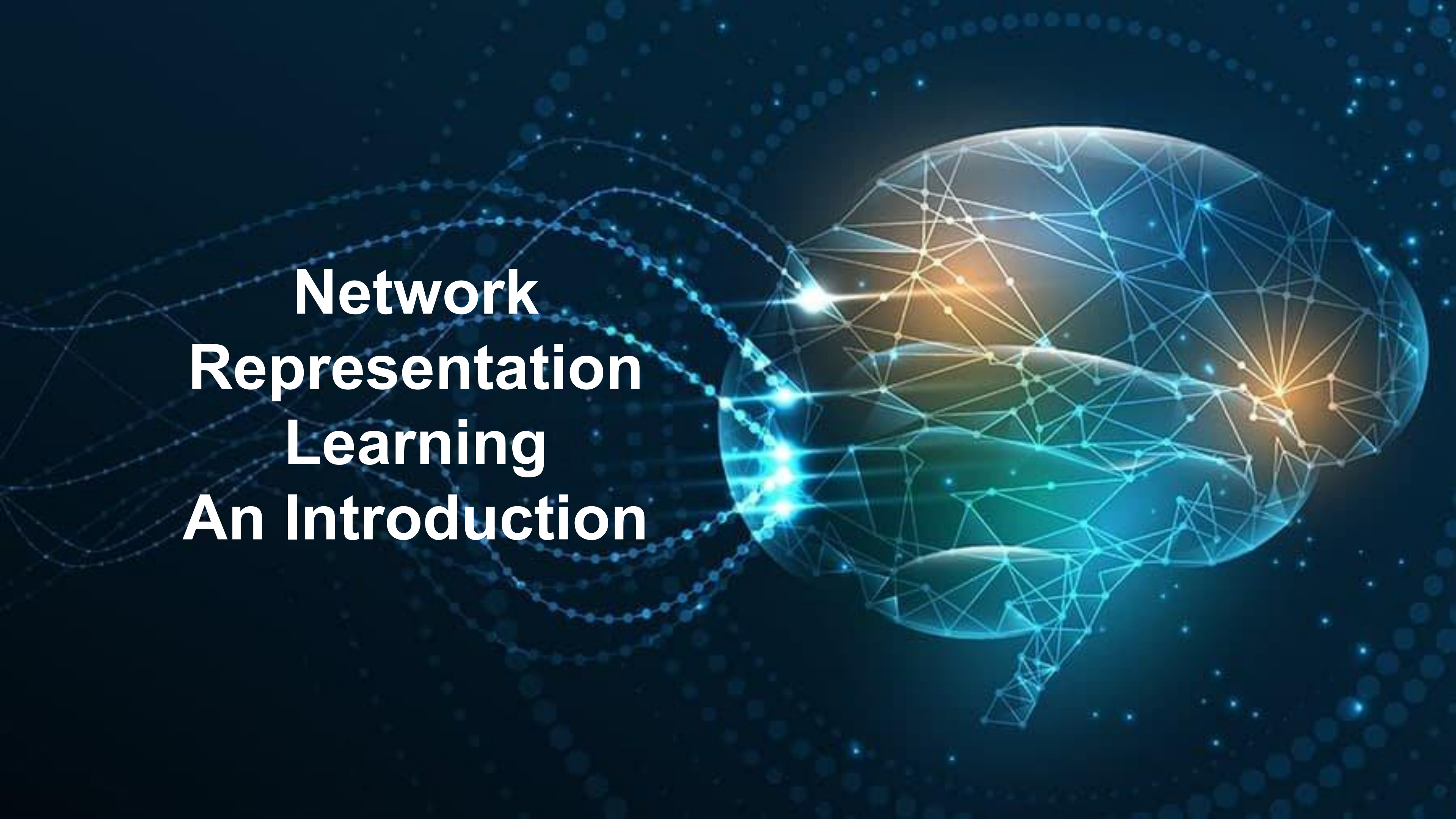
Network Representation Learning An Introduction



Network Representation Learning

Network Representation Learning (NRL) aims to represent the nodes and edges of complex graph structured data into a low dimensional latent space while preserving the network topology, vertex attributes, edge attributes along with some other side information such as vertex labels.

Use of NRL based representations

Representations derived by NRL are used to perform various downstream tasks as follow:

- Clustering
- Link Prediction
- Recommendation Systems
- Vertex Classification

Different NRL Paradigms

To learn the complex graph structure properties, different network paradigms have been proposed.

- ➢ **Matrix Factorization** based methods
  e.g. MNMF (Modularity maximized NMF), HOPE
- ➢ **Random Walk** based methods
  e.g. DeepWalk, Node2Ve
- ➢ **Graph Neural Network** based methods
  e.g. GCN, GraphSage
- ➢ **Transformer** based methods
  e.g. Graph Transformers, Graph-Bert

DeepWalk

A Random Walk Based Method

Ref: https://arxiv.org/pdf/1403.6652.pdf

Introduction

Natural Language Processing has shown significant results by using sentences or continuous sequences of words to train the learning models, e.g. **Skip-gram** or **Continuous Bag of Words** Model. DeepWalk proposes to adapt the methodology in graph context by generating truncated random sequences of nodes by traversing over the edges. This sequences are used to learn the social representations of vertices.

DeepWalk

- ➢ DeepWalk learns the generalized vertex label independent representations of graph nodes, which can be used for different downstream tasks.
- ➢ DeepWalk performs quite well when the graph is sparse which can act as a barrier for performance for others.
- ➢ DeepWalk provides another benefit of being an online algorithm and trivially parallelizable.

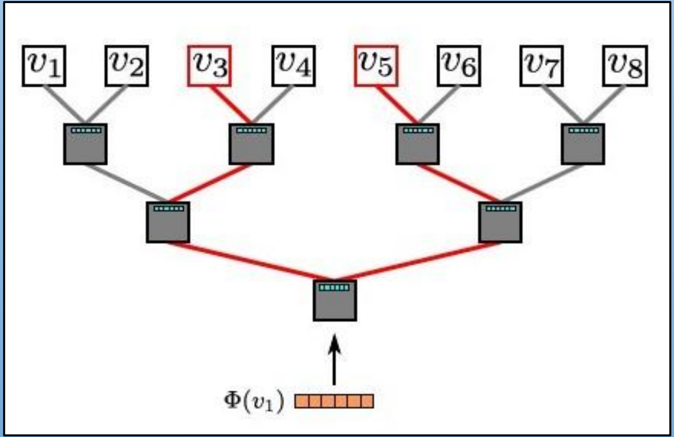
Working of DeepWalk

- ➢ First for any vertex, ***γ*** random walks are generated of length t.
- ➢ Walks starting from a vertex are generated by choosing next vertex uniformly randomly from all neighbours of the current vertex.
- ➢ All this walks are used as context to learn the representations using Skipgram model. Loss function used to maximize the probability of context words:

$$\boxed{\underset{\Phi}{\text{minimize}} \quad -\log \Pr\left(\{v_{i-w}, \cdots, v_{i-1}, v_{i+1}, \cdots, v_{i+w}\} \mid \Phi(v_i)\right)}$$

Working of DeepWalk Continued ...

➢ **Hierarchical Softmax**: In this, all graph nodes are assumed as leaves of a balanced binary tree and all other nodes of the tree act as binary classifiers. Calculating the probability of a vertex u occurring given another vertex v's representation is considered as maximising the probability of path from root to the leaf u,. This reduces the complexity of $Pr(u_k \mid \phi(v))$ from O(|V|) to O(|log(V)|). Now using SGD, parameters of representations are updated.



Future Work

- ➢ Focus on investigating the duality between the graph random walks and word sentences.
- ➢ Make a strong theoretical background for this proposed idea which can provide a good interpretability.
- ➢ Extending it to dynamic graphs and heterogenous graphs.

Node2Vec

A Random Walk Based Method

Ref: https://arxiv.org/pdf/1607.00653.pdf

Introduction

Node2Vec tries to optimize an objective that seeks to preserve the neighbourhood structure from randomly sampled walks like DeepWalk using stochastic gradient descent. But it differs from DeepWalk in the way those sequences are sampled - Node2Vec uses flexible sampling strategy which takes into account both community similarity as well as structural similarity.

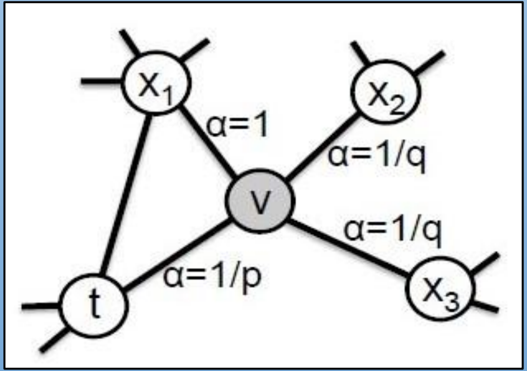
Node2Vec

- ➢ Unsupervised algorithm for scalable feature learning of networks.
- ➢ 2nd order biased random walks are used to explore diverse neighbourhood of a vertex.
- ➢ Using the tunable parameters ***p*** and ***q*** for the sampling can model the full spectrum of equivalences observed in networks.
- ➢ Major phases of Node2Vec are trivially parallelizable.

Tunable Sampling Strategy

- ➢ Prediction task often shuttle between 2 parts: homophily and structural equivalence. BFS is useful for structural equivalence finding while DFS is useful for Homophily finding.
- ➢ **So for a good sampling algorithm, it should have a mixture of both BFS and DFS.**
- ➢ Node2Vec uses a generalized sampling strategy with tunable parameters p and q which can model complete spectrum of equivalences in networks.

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



Working of Node2Vec

It tries to optimise the loss function as shown:
where $N_s(u)$ denotes the context neighbourhood of the vertex u.

$$\max_{f} \sum_{u \in V} \log Pr(N_S(u) | f(u)).$$

This is simplified using 2 assumptions: Conditional Independence and Symmetry in feature space.

So it finally reduces to optimizing the function:
where Z_u is the softmax denominator.

$$\max_{f} \sum_{u \in V} \left[-\log Z_u + \sum_{n_i \in N_S(u)} f(n_i) \cdot f(u) \right]$$

Loss function is optimized using SGD with negative sampling.

Future Work

- ➢ Establish interpretable equivalence notion between parameters and edge representations generated.
- ➢ Extending Node2Vec to heterogenous networks, graphs with domain specific node features and directed networks.

GCN

A Neural Network Based Method

Ref: https://arxiv.org/pdf/1609.02907.pdf

Introduction

Graph Neural Networks take graph data as input to the neural network model and by computing through several hidden layers gives the node embeddings. GCN, a type of graph neural network, averages out the neighbourhood node properties of the node under consideration using stacked convolutional layers. Learning in a semi supervised way, it trains itself on the labelled data available.

GCN

- Semi supervised learning on the part of labelled data and adding the supervised loss in total loss to propagate its affect to the complete network.
- Motivation comes from localised first order approximation of spectral graph convolutions.
- In the cited paper, 2 layers GCN was used.
- Convolutional layers by their construction are highly parallelizable.

Working of GCN

➢ GCN tries to optimize the following loss function:

$$\mathcal{L} = \mathcal{L}_0 + \lambda \mathcal{L}_{\text{reg}}, \quad \text{with} \quad \mathcal{L}_{\text{reg}} = \sum_{i,j} A_{ij} \| f(X_i) - f(X_j) \|^2 = f(X)^\top \Delta f(X)$$

L_0 loss is calculated by the cross entropy loss over all labels for the vertices which are labeled.

$$\mathcal{L} = - \sum_{l \in \mathcal{Y}_L} \sum_{f=1}^{F} Y_{lf} \ln Z_{lf}$$

➢ Adjacency matrix A and node feature matrix X are fed into the network. The propagation rule can be seen in alongside picture.

$$\mathbf{X}^{p+1} = \sigma\left(\tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{X}^p \Theta^p\right),$$

Future Work

- ➢ Graph data has to be stored in memory which acts as a barrier.
- ➢ Improve the GCN with mini batch SGD.
- ➢ Original GCN architecture naturally does not support directed edges or edge features.
- ➢ How much effect the self loop will have compared to neighbourhood edges.
- ➢ Transduction of GCN interferes with the generalization, making the learning of representations of the unseen nodes in the same graph and the nodes in an entirely different graph more difficult.

GraphSAGE

A Neural Network Based Method

Ref: https://arxiv.org/pdf/1706.02216.pdf

Introduction

GraphSAGE is a graph neural network which aggregates the neighbourhood node features of the node under consideration by using different aggregation functions. It is a spatial propagation-based graph convolutional network. It can be trained in a unsupervised or supervised way.

GraphSAGE

- ➢ Provides inductive way to learn the features and perform very well on unknown data and structural groups.
- ➢ Even while it uses feature information to learn, it can learn structural similarity.
- ➢ Works by learning aggregation function at each hidden layer.

Working of GraphSAGE

- ➢ In GraphSAGE, neighbourhood node features are aggregated using aggregation operators instead of having static weight matrices for the purpose. In the original GraphSAGE paper, they used a fixed size set of nodes from the neighbourhood of a node and randomly shuffled them before passing to the aggregation function.
- ➢ It learns k different aggregation functions for k layers of the neural network.
- ➢ Aggregation function should be independent of the order in which nodes are given.

Aggregation Functions

1. **Mean aggregator:** It is different from others in way it joins nodes previous layer embedding to the neighbourhood embeddings for mean.
2. **LSTM aggregator:** It has one problem as LSTM processes input in sequential way so indirectly some sequence structure comes in play. To overcome this, we randomly shuffle the nodes.
3. **Pool aggregator:** It applies non linearity after a fully connected layer and gives aggregator output as the pooling performed over all those results.

Future Work

- ➢ Incorporate directed or multimodal graphs.
- ➢ Using non uniform neighbourhood sampling algorithms.
- ➢ How to include aggregator function as a part of optimization problem.

Attention

It is all you need

Ref: https://arxiv.org/pdf/1706.03762.pdf

Introduction

Attention is mechanism which enables to generalize relations between input and output at global level irrespective of sequence length. Transformer, a new architecture proposed based on attention has outperformed all the existing SOTA architectures. Calculating attention is parallelizable, so giving edge over previous sequential processing of the sequences by RNNs.

Attention

An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.

Calculating Attention

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Here Q is query matrix, K is key matrix and V is values matrix. Dimension of query and key vectors is d_k and dimension of value vector is d_v. Attention functions used is either dot product attention or additive attention. In dot product attention, dot product of query and key is scaled by sqrt(d_k) while in additive attention, they are passed through a feed forward network of single hidden layer.

While for small values of d_k, additive attention has proved to provide better results, but for larger values of d_k, dot product attention is preferred.

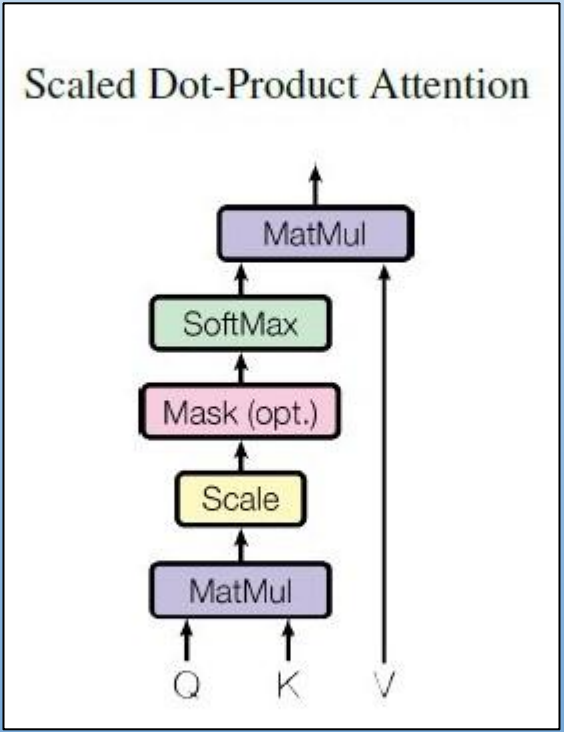
Multi Head Attention

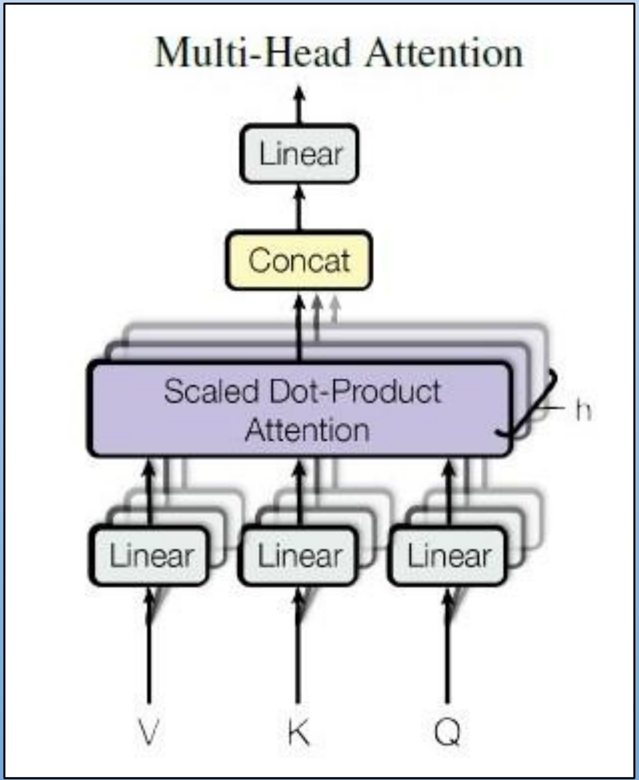
Instead of performing a single attention function with d_{model} dimensional keys, values and queries, it is better to linearly project the queries, keys and values h times with different, learned linear projections to d_k, d_k and d_v dimensions, respectively. On each of these projected versions of queries, keys and values, perform the attention function in parallel, yielding d_v dimensional output values. These are concatenated and once again projected, resulting in the final values.

In this multihead setting, $d_k = d_v = d_{model} / h$

$$\begin{aligned} \mathrm{MultiHead}(Q, K, V) &= \mathrm{Concat}(\mathrm{head}_1, ..., \mathrm{head}_\mathrm{h})W^O \\ \text{where } \mathrm{head}_\mathrm{i} &= \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V) \end{aligned}$$

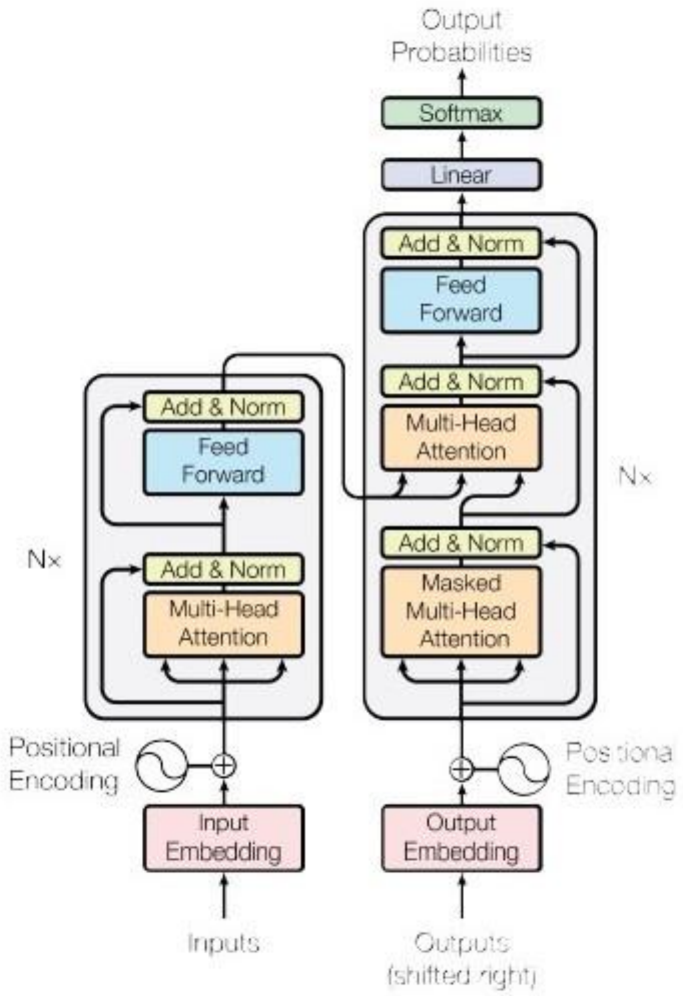
Attention Calculation (Diag)





Transformer Architecture

There are N stacked units of each at encoder as well as decoder part.



Positional Encoding

When attention is calculated, there is no sequential order is considered. So to enforce the sequence information, positional encoding is added to the embeddings fed into model.

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

Graph Transformer

Transformer Based Method

Ref: https://arxiv.org/pdf/2012.09699.pdf

Introduction

Graph Transformer uses transformer based architecture on graph data along with positional encoding to generate inductive model. It calculates attention over neighbourhood of the node under consideration rather than all nodes like sentence. It uses graph sparsity efficiently for attention calculation. Due to its inductive nature, generalizes well to unknown nodes and datasets.

Graph Transformer

- Sparsity and positional encodings are two key aspects in the development of a Graph Transformer.
- Works on arbitrary sized homogeneous graphs and can be extended to consider edge features as well.
- Uses laplacian eigenvectors as positional embeddings for nodes.
- Uses node features to learn representation.

Working of Graph Transformer

- ➢ First node input features and node embeddings to be passed to the Graph Transformer are prepared by adding linear transformation of node features and positional embedding to k dimensional space.
- ➢ k smallest non-trivial eigenvectors of graph laplacian are used as positional embeddings for nodes.
- ➢ To calculate the next layer representation, multi head attention of node in consideration is calculated by taking attention with nodes in neighbourhood.

Working of Graph Transformer Continued ...

$$\hat{h}_i^{\ell+1} = O_h^{\ell} \mathop{\Big\Vert}_{k=1}^{H} \Big(\sum_{j \in \mathcal{N}_i} w_{ij}^{k,\ell} V^{k,\ell} h_j^{\ell} \Big),$$

$$\text{where, } w_{ij}^{k,\ell} = \text{softmax}_j \Big(\frac{Q^{k,\ell} h_i^{\ell} \cdot K^{k,\ell} h_j^{\ell}}{\sqrt{d_k}} \Big)$$

$$\hat{\hat{h}}_i^{\ell+1} = \text{Norm}\Big(h_i^{\ell} + \hat{h}_i^{\ell+1} \Big),$$

$$\hat{\hat{\hat{h}}}_i^{\ell+1} = W_2^{\ell} \text{ReLU}(W_1^{\ell} \hat{\hat{h}}_i^{\ell+1}),$$

$$h_i^{\ell+1} = \text{Norm}\Big(\hat{\hat{h}}_i^{\ell+1} + \hat{\hat{\hat{h}}}_i^{\ell+1} \Big)$$

This equations are transformer equations used to calculate values across layers in network. The representation at last layer are passed through MLP layer for task dependent learning and loss function.

Future Work

- Efficient training on single large graphs
- Application of transformers in heterogeneous domain
- Use recent advancements of inductive biases in graph representation learning

Graph BERT

Transformer Based Method

Ref: https://arxiv.org/pdf/2001.05140.pdf

Introduction

Graph BERT uses graph data along with positional encoding to generate inductive model. It calculates attention over neighbourhood of the node under consideration rather than all nodes like sentence. It uses graph sparsity efficiently for attention calculation. Due to its inductive nature, generalizes well to unknown nodes and datasets.

Graph BERT

- ➢ Training is independent of the links in the graph, making it able to learn more generalized representations.
- ➢ Training is done in unsupervised way.
- ➢ This model can be transferred with some necessary fine tuning for some specific downstream task which is not possible in most models of GNN
- ➢ Learning is done on sampled subgraphs instead of the original graph.

Precomputed Embeddings

4 types of embeddings are fed into model for training:

1. Raw Feature Vector Embedding: embed node's raw feature vector into a shared feature space by Embed function.

$$\mathrm{e}_j^{(x)} = \mathrm{Embed}\left(\mathbf{x}_j\right) \in \mathbb{R}^{d_h \times 1}.$$

2. Weisfeiler-Lehman Absolute Role Embedding: Weisfeiler-Lehman (WL) algorithm, nodes are labelled value according to their structural roles in the graph data

$$\begin{aligned}\mathrm{e}_j^{(r)} &= \mathrm{Position\text{-}Embed}\left(\mathrm{WL}(v_j)\right)\\ &= \left[sin\left(\frac{\mathrm{WL}(v_j)}{10000^{\frac{2l}{d_h}}}\right), cos\left(\frac{\mathrm{WL}(v_j)}{10000^{\frac{2l+1}{d_h}}}\right)\right]_{l=0}^{\left\lfloor \frac{d_h}{2} \right\rfloor},\end{aligned}$$

Precomputed Embeddings

3. Intimacy based Relative Positional Embedding: nodes in the sampled graph are ordered in the decreasing intimacy score and this position is used to define this encoding by

$$\mathrm{e}_j^{(p)} = \text{Position-Embed}\left(\mathrm{P}(v_j)\right) \in \mathbb{R}^{d_h \times 1},$$

4. Hop based Relative Distance Embedding: for node $v_j \in V_i$ in the subgraph g_i, we can denote its relative distance in hops to v_i in the original input graph as $H(v_j\ ;\ v_i)$. Then

$$\mathrm{e}_j^{(d)} = \text{Position-Embed}\left(\mathrm{H}(v_j; v_i)\right) \in \mathbb{R}^{d_h \times 1}.$$

Graph BERT Working

- Initial h_j^0 is defined for node v_j in subgraph g_i

$$h_j^{(0)} = \text{Aggregate}(e_j^{(x)}, e_j^{(r)}, e_j^{(p)}, e_j^{(d)}).$$

  Initial input vectors of all nodes $H^{(0)}$ in subgraph g_i is defined by contcataning all the h vectors corresponding to nodes in subgraph g_i.
- This H value is given to model and further layers are calculated as shown aside.
- At the last layer, fusion function Is applied which averages out embeddings of all nodes in g_i to Get representation of V_i.

$$\begin{aligned}\mathbf{H}^{(l)} &= \text{G-Transformer}\left(\mathbf{H}^{(l-1)}\right)\\ &= \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_h}}\right)\mathbf{V} + \text{G-Res}\left(\mathbf{H}^{(l-1)}, \mathbf{X}_i\right),\end{aligned}$$

where

$$\begin{cases}\mathbf{Q} &= \mathbf{H}^{(l-1)}\mathbf{W}_Q^{(l)},\\ \mathbf{K} &= \mathbf{H}^{(l-1)}\mathbf{W}_K^{(l)},\\ \mathbf{V} &= \mathbf{H}^{(l-1)}\mathbf{W}_V^{(l)}.\end{cases}$$

Future Work

- Application in heterogeneous domain

Questions?

Thank You !!