Final INTERNSHIP REPORT:
IISC BANGALORE
Guide: Dr. Sundeep Prabhakar Chepuri

Candidate Name: Ananya Singha
Application no: FENGS51

Graph Neural Network and it's Application

ABSTRACT:

Graphs, which describe pairwise relations between entities, are essential representations for real-world data from many different domains. Hence, research on graphs has attracted immense attention from multiple disciplines. In order to study graph, we first do graph embedding which helps to represent graph nodes, edges, or subgraphs in low-dimensional vectors. **Deep walk** was the first representation learning inspired model used for graph embedding using random walk along with skip-gram to collect information around the graph.
Due to the transductive nature and higher computational need being the drawback of deep walk, graph neural network gained popularity. **GNN** are neural networks that operate on the structured graph to extract node embedding and help to solve problems like link prediction, node classification, cluster detection etc. **GCN(Graph Convolution Network)** uses convolution layers inspired by the CNN to do graph embedding. Just like CNN kernels/filters perform convolution over the neighbouring nodes of a given node. GCN has several invariants like GAT, GIN, Gsage each of them differ from each other based on their message computation and aggregation methods. **Gsage** offers a variety of choices for aggregation methods like mean, LSTM, MAXpooling along with concatenation of self-embedding from the previous layer, which prevents information loss. **GAT** uses an attention module that helps it to prioritize important neighbours information unlike GCN and Gsage which give equal priority to all neighbours. **GIN** is called the most expressive as it preserves essence of the graph structure because of its robust message computation method where it uses a special function to capture the structural information of the graph.To see the performance of GCN, Gsage, GAT on node classification and link prediction task I used dgl library and Cora graph dataset. **Results were as aspected GAT outperformed with accuracy of 78.3% (node classification) and AUC of 0.92(link prediction) followed by Gsage with the accuracy of 76.9% (node classification) and AUC of 0.88(link prediction) then GCN with the accuracy of**

76.8% (node classification) and AUC of 0.87(link prediction). In the coming week, we will be using **GNN to solve the graph entity alignment problem,** given two knowledge graphs(KG)we will try to find if there exist any links between these two KGs.The methodology and results are described in section 2.1

1 INTRODUCTION:

1.1 GRAPHS:

A graph is a data structure consisting of two components, vertices and edges. A graph G can be well described by the set of vertices V and edges E it contains. Edges can be either directed or undirected, depending on whether there exist directional dependencies between vertices(nodes).

1.2 GRAPH NEURAL NETWORK:

Graph Neural Network is a type of Neural Network which directly operates on the Graph structure(non-euclidean data). It helps to represent the non-euclidean data into a Euclidean space for better handling. The Euclidean space representation of these nodes with their characteristic is called graph embeddings. These graph embeddings are then used for different problem solving like:

- Node classification
- Link prediction
- Graph classification
- Community Detection, etc.

1.3 DEEP WALK-a graph embedding method - Perozzi et al.

Deep walk belongs to the family of graph embedding techniques which uses walks to aggregate node representation of neighbours guided by an adjacency matrix.
The main idea is to find an embedding of nodes in d dimensional space that preserves similarity. For the same, The goal is to estimate the likelihood of observing node ***vi*** given all the previous nodes visited so far in the random walk, where ***Pr()*** is a probability, a mapping function Φ is used

$$\Phi : v \in V \mapsto \mathbb{R}^{|V| \times d}.$$

$\Pr\left(v_i \mid \left(\Phi(v_1), \Phi(v_2), \cdots, \Phi(v_{i-1})\right)\right)$ with optimization of

$\underset{\Phi}{\text{minimize}} \quad -\log \Pr\left(\{v_{i-w}, \cdots, v_{i-1}, v_{i+1}, \cdots, v_{i+w}\} \mid \Phi(v_i)\right)$ which helps to capture similarities between the nodes.
The above optimization is known as skip gram which is a language model that maximizes the co-occurrence probability among the words that appear within a window, w, in a sentence. They also use hierarchical softmax rather than softmax to speed up the training.

Although DeepWalk is relatively efficient with a score of **O(|V|)***,* this approach is **transductive**, meaning whenever a new node is added, the model must be retrained to embed and learn from the new node.

Next the most recent methods used are **GRAPH CONVOLUTIONS**:
There are different types of graph convs, each of them mainly differ from one another in two main domain

- Message computation
- Aggregation

The above two functions are what makes every GNN unique.
The message computation is the first step whose goal is to pass the message of a neighbouring node to the next node. It can be simply done as an average or can be highly complex and computational like using MLP.

The Aggregation is the second step which deals with the aggregation of the message passed by the neighbouring nodes. It can be max, min or sum.

The second step can proceed further can use the concatenation of aggregated neighbour information along with current self-information. Which makes the model more informative and expressive.

The training and optimization for all these networks remain similar.

They also help to mitigate the aforementioned problem, learning the embedding for each node in an inductive way. Specifically, each node is represented by the aggregation of its neighbourhood. Thus, even if a new node unseen during training time appears in the graph, it can still be properly represented by its neighbouring nodes.

1.4 GRAPH CONVOLUTION NETWORK(GCN):

Among the most cited works in graph learning is a paper by Kipf and Welling. The paper introduced spectral convolutions to graph learning, and was dubbed simply as “graph convolutional networks”, which is by no means the origin of all subsequent works in graph learning. In Kipf and Welling's GCN, a layer-wise graph convolution is defined by:

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right).$$

Where H is the node embeddings. D is the diagonal Matrix with node degree as their entry. A is the adjacency matrix. W is the weight matrix. The same in vector form can be represented as:

$$h_{v_i}^{(l+1)} = \sigma\left(\sum_j \frac{1}{c_{ij}} h_{v_j}^{(l)} W^{(l)}\right)$$

This is how the message and aggregation parts are for GCN.

1.5 GRAPH SAGE:

Graph sage differs from GCN in terms of expressiveness and the Aggregation function. As in the previous layer embedding of the same node was ignored. But Graph sage takes care of the embedding of the same node from the previous layer by concatenating it with the neighbouring nodes line 5 in the pseudo code.
Below is the algorithm for Graph Sage:

```
Algorithm 1: GraphSAGE embedding generation (i.e., forward propagation) algorithm
    Input  : Graph $\mathcal{G}(\mathcal{V},\mathcal{E})$; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; depth $K$; weight matrices
             $\mathbf{W}^k, \forall k \in \{1,...,K\}$; non-linearity $\sigma$; differentiable aggregator functions
             $\text{AGGREGATE}_k, \forall k \in \{1,...,K\}$; neighborhood function $\mathcal{N}: v \to 2^{\mathcal{V}}$
    Output : Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{V}$
1 $\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$ ;
2 for $k = 1...K$ do
3     for $v \in \mathcal{V}$ do
4         $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;
5         $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$
6     end
7     $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$
8 end
9 $\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$
```

The paper also choices of different aggregators:

- **Mean aggregator**

$$\mathbf{h}_v^k \leftarrow \sigma(\mathbf{W} \cdot \text{MEAN}(\{\mathbf{h}_v^{k-1}\} \cup \{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\}).$$

- **LSTM aggregator**
- **Max pooling**

$$\text{AGGREGATE}_k^{\text{pool}} = \max(\{\sigma\left(\mathbf{W}_{\text{pool}}\mathbf{h}_{u_i}^k + \mathbf{b}\right), \forall u_i \in \mathcal{N}(v)\}),$$

1.6 GRAPH ATTENTION NETWORK(GAT):

The main idea is to leverage the existing GCN with a masked self-attention layer.

By stacking layers in which nodes are able to attend over their neighborhoods' features, it implicitly specifying different weights to different nodes in a neighborhood,unlike GRAPH SAGE and GCN which gives equivalent importance to each node. a layer-wise graph convolution is defined by

$$\vec{h}'_i = \sigma\left(\sum_{j\in\mathcal{N}_i} \alpha_{ij}\mathbf{W}\vec{h}_j\right)$$

where α_{ij} is the attention weights

$$e_{ij} = a(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j)$$

where the attention mechanism a is a single-layer feedforward neural network.

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_j]\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_k]\right)\right)}$$

where · T represents transposition and || is the concatenation operation.

1.7 Experiments:
I have applied **GCN, GRAPH SAGE, GAT** for node classification and link prediction.

1.7.1 NODE CLASSIFICATION:
Concretely, Node Classification models are used to predict a non-existing node property based on other node properties. The non-existing node property represents the class, and is referred to as the target property. The specified node properties are used as input features.
Dataset used:
Cora graph(citation network) dataset from dgl dataset library.
Details about the dataset is given below:
Nodes mean paper and edges mean citation relationships. Each node has a predefined feature with 1433 dimensions. The dataset is designed for the node classification task. The task is to predict the category of certain paper.

Statistics:

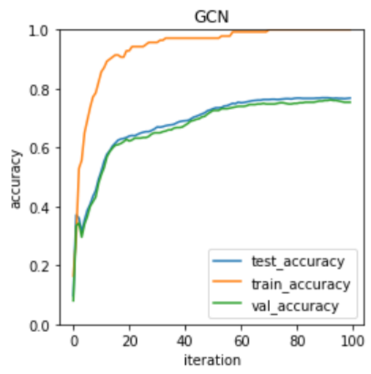
- Nodes: 2708
- Edges: 10556
- Number of Classes: 7
  - Label split:
    - Train: 140
    - Valid: 500
    - Test: 1000

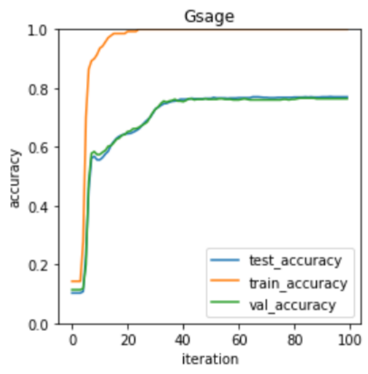
Data preprocessing steps include:

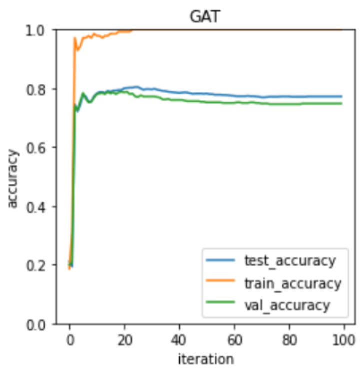
1. Dividing the data into train, validation and test data.
2. Mode is trained on trained data and tested on test data

Each nn model has 3 layers with one hidden layer of 16 neurons, between each layer convolution is applied followed by RELU activation.. Binary Cross entropy loss along with adam optimizer is used to optimize and upgrade the weights.

The performance of different models are shown below after running for 100 epochs.







Val_accuracy: 76.2% **Val_accuracy**: 76.6%
Val_accuracy:77%
Test_accuracy: 76.8% **test_accuracy:** 76.9%
test_accuracy:78.3%

GAT outperform other models as we can see from the above table, which is not at all surprising as it is more expressive than GCN and Gsage

1.7.2 Link prediction:

Link prediction is the problem of predicting the existence of a link between two entities in a network.

Dataset used :
Cora graph dataset from dgl library. Same as used in node classification.

Preprocessing of data:

- Data is divided into a train and test data.(they consist of paired nodes(u,v) which are incoming and outgoing nodes of edges.
- Further train is divided into train_pos and train_neg where train_pos means graph which has edges with them(aka linked) , similarly train_neg means graph which has no edge in them(aka not linked). Same is done for test data as well.

Each model has 3 layers with a hidden layer of 16 neurons and between each layer a convolution is performed followed by RELU activation. The graph along with the activation from the model is passed to A dot predictor to predict the scores. When is then used to compute loss and optimized using adams optimizer.
Below the performance of each model is given in terms of AUC:
AUC is a stronger metric than accuracy because on imbalanced dataset using the majority run as a classifier will lead to high accuracy which will make it a misleading measure. AUC aggregate over confidence threshold, and thus is more accurate.
The performance of different models are tabulated below after running for 100 epochs.

MODEL	**GCN**	**GRAPH SAGE**	**GAT**
AUC score	0.87	0.88	0.92

GAT outperform other models as we can see from the above table, which is not at all surprising as it is more expressive than GCN and Gsage

2.Entity alignment

2.1 Knowledge Graph
A knowledge graph (e.g. Freebase, YAGO) is a multi-relational graph representing rich factual information among entities of various types. Knowledge Graphs (KGs) contain a large volume of relation tuples in the form of ⟨subject, relation, object. These relation tuples have a variety of downstream applications including Question Answering, Search, and Recommendation. With the booming structured and semi-structured online data, numerous knowledge graphs are extracted on the same domain. Different KGs, though subject to incompleteness in varying degrees, usually contain complementary information.

2.2 Entity alignment
With several KG comes the task of entity alignment which aims to identity entities across different KG which are potentially the same element in real life.

Definition 2.1 (KG Entity Alignment). Given two knowledge graphs G = (V,E,R) and G′ = (V′,E′R), entity alignment aims to find a set of entity pairs {(vi,vi′)∈V×V }with high precision and recall, such that each pair refers to the same real-world entity.

Our goal in the project is to do entity alignment as described above.
For motivation, we read the paper “Collective Multi-type Entity Alignment Between Knowledge Graphs”.

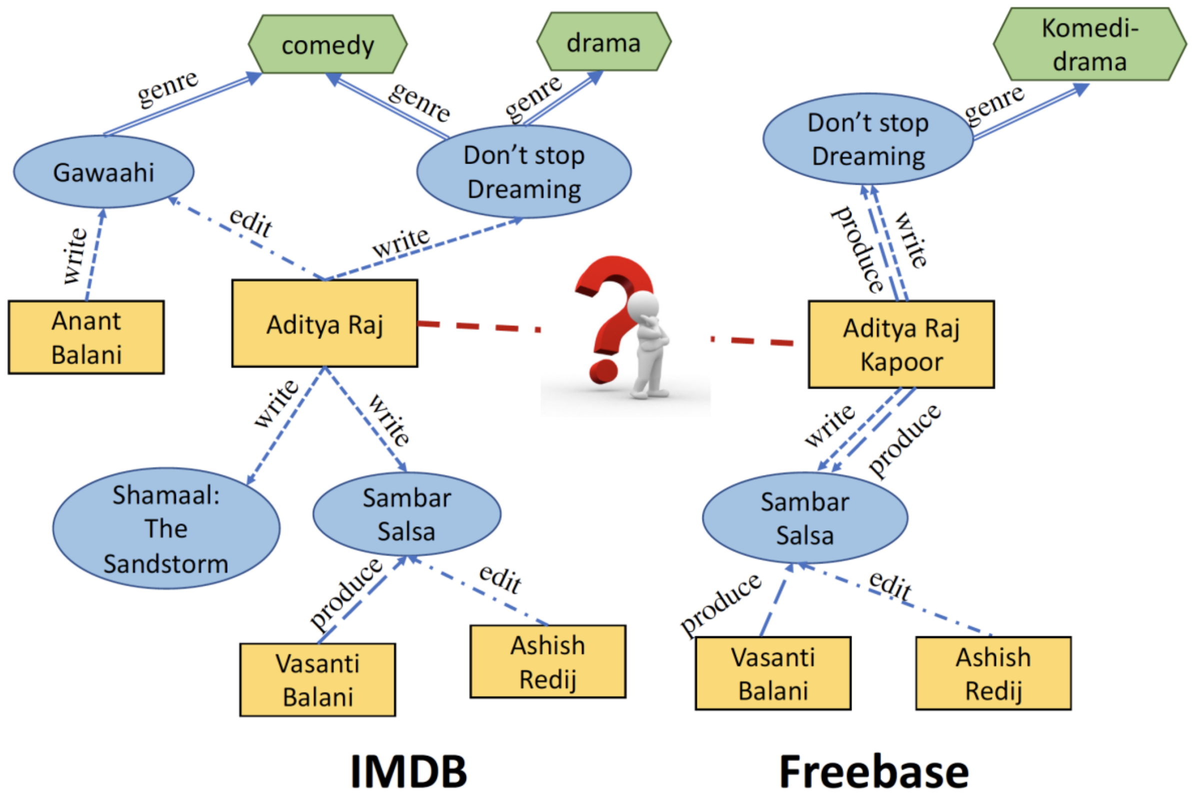


FIG: the above Image describing the task of entity alignment taken from paper **'Collective knowledge graph multi-type entity alignment'**

2.3 RELATED WORK:

The earliest usage of graph neural network for entity alignment is **GCN-Align**. GCN-Align uses GCN to learn the vector representation of entities, and at the same time, allows two GCNs encoding different knowledge graphs to use the same parameters so that they can use the structure between entities to propagate the alignment relationship between entities.Although the graph neural network is used to encode the information of the graph, the current model only considers node-level alignment. One of the recent work in has been **CG-MuAlign** which uses the self-attention mechanism between entities and relationships to improve the computational efficiency of the method in large-scale data collection. Our work is inspired by **CG-MuAlign** where they use two GNN with similar architecture to capture the similarities between the KGs, and then uses distance based method to compute the similarity score along with marginal hinge loss and Adam optimiser to update and optimise the model.**HyperKA** extends translational and GNN-based techniques to hyperbolic spaces, and captures associations by a hyperbolic transformation. This work introduces the entity representation of knowledge graphs in non-Euclidean space into entity alignment tasks for the first time, and it performs well in low-dimensional settings. We believe that the expressive power of hyperbolic space is fundamentally different from that of traditional Euclidean space. Fully exploring the performance of non-Euclidean space in entity alignment tasks will be an exciting research direction in future work.

2.4 METHODOLOGY:

Our method is inspired by the **siamese network**. Where given two images we pass them through the same CNN network to get the feature vector representing each image and then compute the similarity score to group a similar set of images.

Similarly in our case, we had two KGs whose entries were to be aligned.

Given $KG_1(V_1, E_1, T_1, R_1)$ and KG_2 (V_2, E_2, T_2, R_2) we pass it through a deep graph network to compute the node embeddings for graphs e_1, e_2, respectively. These embeddings are then passed through a function to compute the similarity score (s) between the nodes, which helps to identify the entity pairs $\{(vi_1,vi_2)\in V_1 \times V_2\}$ as defined by definition 1.1.

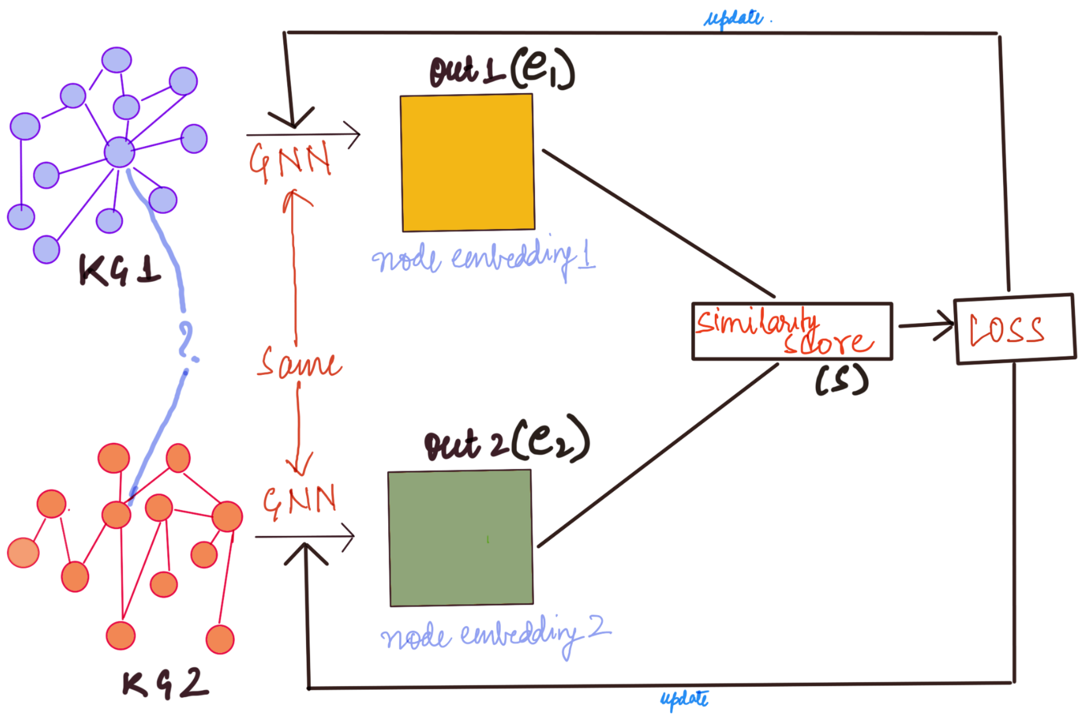
$\mathbf{e_1 = f(KG_1(V_1, E_1, T_1, R_1))}$ where d is the length of the feature vector and e_1 $R^{V1 \times d}$

$\mathbf{e_2 = f(KG_2\ (V_2, E_2, T_2, R_2))}$ where d is the length of the feature vector and e_2 $R^{V2 \times d}$

$\mathbf{s = g(e_1,e_2)}$ where s $R^{V \times 1}$ where V is the no of training samples

y = 0 if s >=0 else 1

Here f is a neural network and g is the similarity score function. And the output y is used to calculate accuracy and f1 score.



Further, the network is optimized using adam optimizer and binary cross-entropy loss with logits(BCE with logits). The code can be seen via this link()

Hyperparameter tuning is done along with early stopping with help of a validation set.

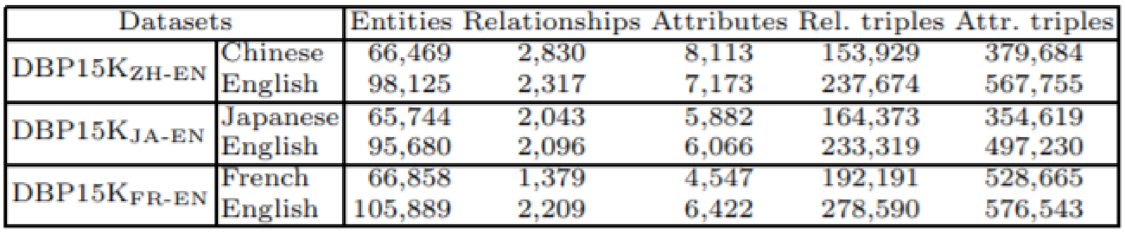
F1 score, AUC and accuracy are determined for the model to understand their performance.

In order to access the performance of different deep networks on our described methodology, we choose MLP, Convolution network and 3 graph networks. The performance of each network and their comparison is done in the Result section.

2.5 DATASET:

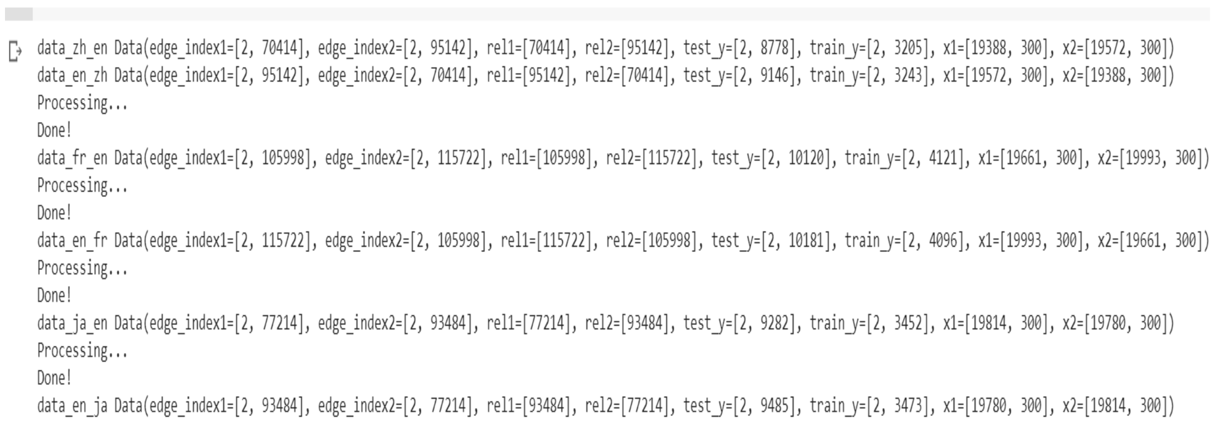
We are using the DBP15k dataset from PyTorch geometric dataset, which comes in a preprocessed manner. The dataset is based on multilingual data. They have entities in english and their respective names in other language. The dataset has 3 different pairs of graphs as described below:

Table 1: Statistics of the datasets



Datasets		Entities	Relationships	Attributes	Rel. triples	Attr. triples
DBP15K$_{ZH\text{-}EN}$	Chinese	66,469	2,830	8,113	153,929	379,684
	English	98,125	2,317	7,173	237,674	567,755
DBP15K$_{JA\text{-}EN}$	Japanese	65,744	2,043	5,882	164,373	354,619
	English	95,680	2,096	6,066	233,319	497,230
DBP15K$_{FR\text{-}EN}$	French	66,858	1,379	4,547	192,191	528,665
	English	105,889	2,209	6,422	278,590	576,543

They also provide the permuted pair of the dataset pairs.
The pic shows how the dataset is given inside the PyTorch graph data module for all 6 pairs.



```
data_zh_en Data(edge_index1=[2, 70414], edge_index2=[2, 95142], rel1=[70414], rel2=[95142], test_y=[2, 8778], train_y=[2, 3205], x1=[19388, 300], x2=[19572, 300])
data_en_zh Data(edge_index1=[2, 95142], edge_index2=[2, 70414], rel1=[95142], rel2=[70414], test_y=[2, 9146], train_y=[2, 3243], x1=[19572, 300], x2=[19388, 300])
Processing...
Done!
data_fr_en Data(edge_index1=[2, 105998], edge_index2=[2, 115722], rel1=[105998], rel2=[115722], test_y=[2, 10120], train_y=[2, 4121], x1=[19661, 300], x2=[19993, 300])
Processing...
Done!
data_en_fr Data(edge_index1=[2, 115722], edge_index2=[2, 105998], rel1=[115722], rel2=[105998], test_y=[2, 10181], train_y=[2, 4096], x1=[19993, 300], x2=[19661, 300])
Processing...
Done!
data_ja_en Data(edge_index1=[2, 77214], edge_index2=[2, 93484], rel1=[77214], rel2=[93484], test_y=[2, 9282], train_y=[2, 3452], x1=[19814, 300], x2=[19780, 300])
Processing...
Done!
data_en_ja Data(edge_index1=[2, 93484], edge_index2=[2, 77214], rel1=[93484], rel2=[77214], test_y=[2, 9485], train_y=[2, 3473], x1=[19780, 300], x2=[19814, 300])
```

We apply our proposed model to all 6 pairs of the graph dataset to see how performance vary for the different graph dataset.

We divided the dataset into train and test according to the given test and train set of the respective dataset. We further divided the test set into two halves for making a validation set out of it.

2.6 DEEP NETWORKS :

2.6.1 MLP:

We use multi-layer perceptions which are composed linear NN. we stack two layer of linear NN with an input layer of 300 neurons and a hidden layer of 150 neurons and final layer with 7 neurons. Which is then passed forward with ReLu activation function.

We stopped the training at 65th iter to avoid overfitting. The result is mentioned below:

```
In epoch 65, loss: 0.32536521553993225,
train_accuracy:0.9911111111111112, val_accuracy:
0.8232044198895028
--------------------------------------------------
test_auc_score 0.8815227951305356
test_accuracy 0.8304761904761905
test_f1_score 0.8951954780970325
```

Total no of parameters in the model:

```
model_summary

Layer_name                                Number of Parameters
==================================================================
==================================



Linear(in_features=300, out_features=150, bias=True)      45000

Linear(in_features=150, out_features=7, bias=True)
150
==================================================================
==================================
Total Params:45150
None
```

2.6.2 Convolution neural network:

Since the feature vector given to us is a 2 dimensional we use conv1d for this purpose. We stacked 2 layers of Conv1d with a kernel size of 3 and stride 1 for both the layers. The output has a vector dimension of length 293. We pass these layers forward with Relu activation.

We stopped the training after 25th iter to avoid overfitting as guided by our validation set.

Results are shown below:

```
In epoch 25, loss: 0.5544642210006714,  train_accuracy:
0.7122222222222222, val_accuracy: 0.8367308058677844
--------------------------------------------------
test_auc_score 0.9683637729780321
test_accuracy 0.8352380952380952
test_f1_score 0.9102231447846394
```

Total no of parameters in the model:

```
model_summary

Layer_name                                Number of Parameters
==================================================================
==================================


Conv1d(1, 1, kernel_size=(3,), stride=(1,))               3

Conv1d(1, 1, kernel_size=(3,), stride=(1,))               1
==================================================================
==================================
Total Params:4
None
```

2.6.3 Graph Convolution network:

graph convolution networks are defined in sections. Here we use two graph conv layers with an input layer of 300 neurons and hidden layer of 150 neurons and the final layer has 7 neurons. Similar to the previous network we use Relu activation in the forward pass.

We stopped the training after 10 th iter to avoid overfitting as guided by our validation set.
Results are shown below:

```
In epoch 10, loss: 0.496075838804245,  train_accuracy: 
0.8242222222222222, val_accuracy: 0.8304438940750619
---------------------------------------------------
test_auc_score 0.7669290342141166
test_accuracy 0.8220952380952381
test_f1_score 0.8948198198198197
```

Total no of parameters in the model:

```
MODEL: GRAPH CONV(GCN)
model_summary




Layer_name                                Number of Parameters
==================================================================
==================================

GCNConv(300, 150)              45000

GCNConv(150, 7)           150
==================================================================
==================================
Total Params:45150
```

2.6.4 Graph sage:
This model consists of two graph sage layers defined in section   with input layer having 300 neurons and one hidden layer with 150 neuron and final layer of 7 neurons. We again use the same ReLu action because its mathematical robustness is avoiding diminishing gradients in the forward pass.

We stopped the training after 15th iter to avoid overfitting as guided by our validation set.
Results are shown below:

```
In epoch 15, loss: 0.40843847393989563,  train_accuracy: 
0.9206666666666666, val_accuracy: 0.8672128024385597
```

```
--------------------------------------------------
test_auc_score 0.9016181016471023
test_accuracy 0.8695238095238095
test_f1_score 0.9211465408081041
```

Total no of parameters in the model:

```
model_summary

Layer_name                                Number of Parameters
==================================================================
==================================

SAGEConv(300, 150)             45150

SAGEConv(150, 7)               46050
==================================================================
==================================
Total Params:91200
```

2.6.5 Graph Attention network:
Similar to previous networks , here, we also use only two layers of GAT layers from PyTorch geometric library `GATConv with` input layer having 300 neurons and one hidden layer with 150 neurons and the final layer of 7 neurons. The same Relu activation function is used in the forward pass.

We stopped the training after 15th iter to avoid overfitting as guided by our validation set.
Results are shown below:

```
In epoch 15, loss: 0.44742608070373535,  train_accuracy:
0.8906666666666667, val_accuracy: 0.8003429224614212
--------------------------------------------------
test_auc_score 0.7520821771541184
test_accuracy 0.7881904761904762
test_f1_score 0.8716528162511542
```

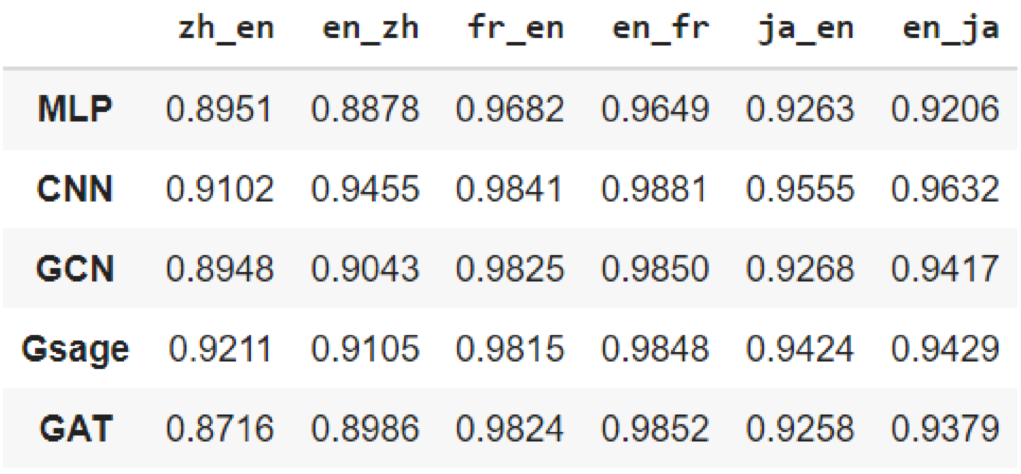
Total no of parameters in the model:

```
MODEL:  GAT
model_summary

Layer_name                                Number of Parameters
==================================================================
==================================

GATConv(300, 150, heads=1)          150
GATConv(150, 7, heads=1)              150
===================================================================
==================================
Total Params:300
```

We have kept the architecture of these 5 networks similar as far as possible so that we can compare their performance.

2.7 RESULTS:

In order to study which model performs better which dataset and compare performance, we used the f1 score to prepare the table below which shows how performance varies among different datasets and neural networks.



	zh_en	en_zh	fr_en	en_fr	ja_en	en_ja
MLP	0.8951	0.8878	0.9682	0.9649	0.9263	0.9206
CNN	0.9102	0.9455	0.9841	0.9881	0.9555	0.9632
GCN	0.8948	0.9043	0.9825	0.9850	0.9268	0.9417
Gsage	0.9211	0.9105	0.9815	0.9848	0.9424	0.9429
GAT	0.8716	0.8986	0.9824	0.9852	0.9258	0.9379

TABLE: Report the f1 score of different network architectures on different datasets for our proposed methodology.

–

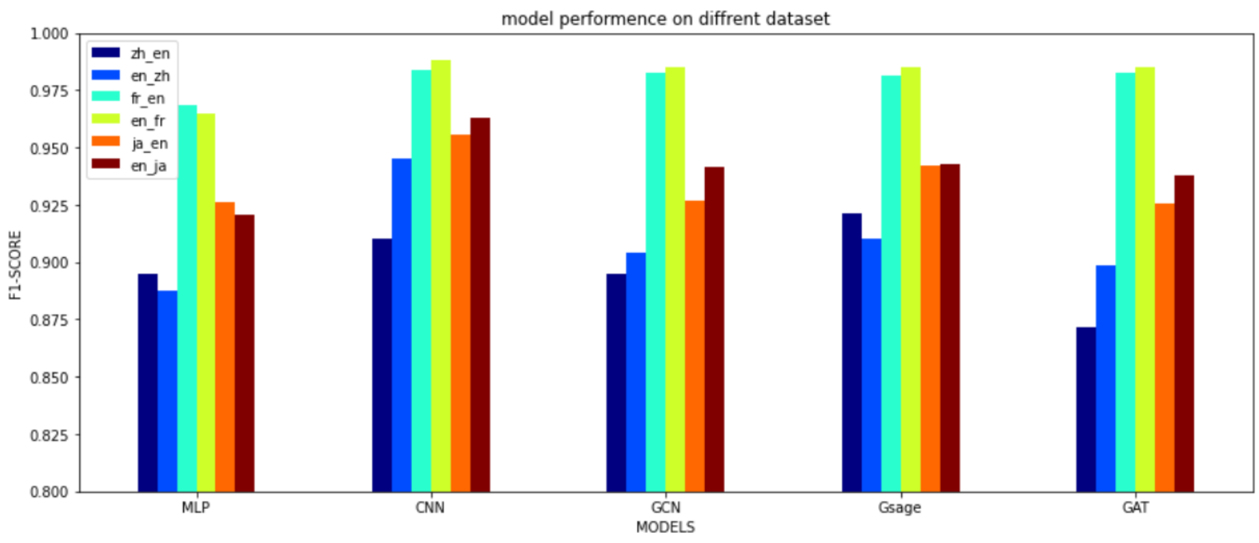


FIG: Bar graph for better visualization of the table above.

In the fig above Y-axis report the f1 score of the models on different datasets, X-axis denotes different neural networks and each model has 6 bar each for different pair of the data. F1 score signifies how accurately the entity alignment was score nearer to 1 means the performance was good. The legends gives detailed about the graph pairs which are 6 in total: zh-en(Chinese-English), en-zh(English-Chinese), fr-en(french-English), en-fr(English- french), ja-en(Japanese- English), en-ja(English - Japanese).

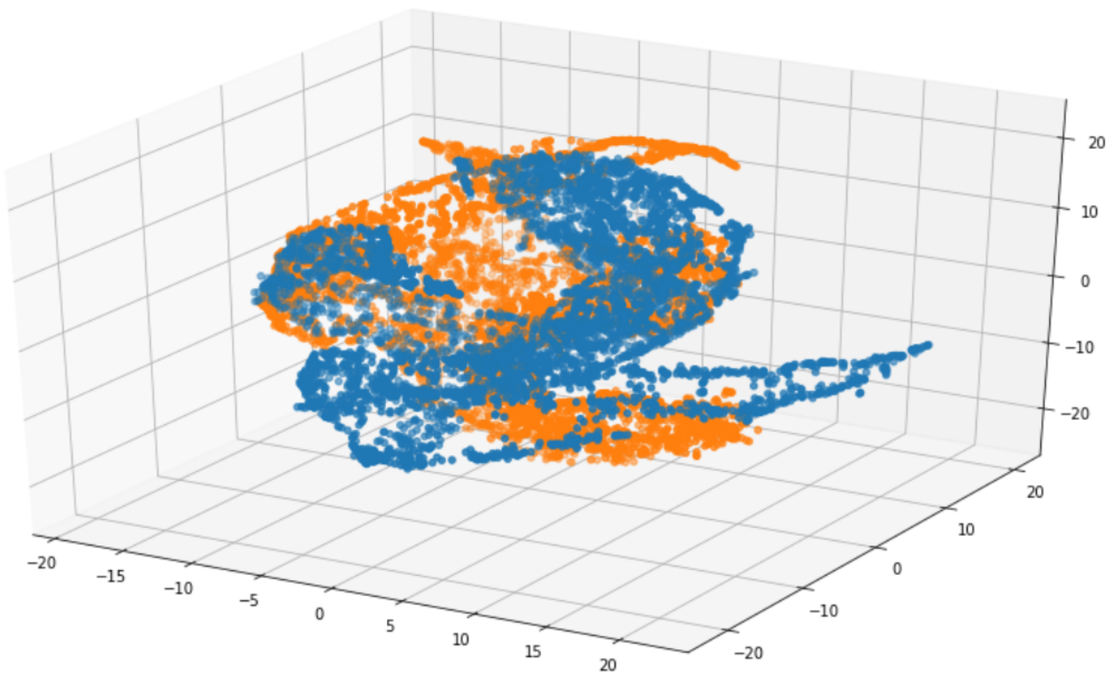


Fig: shows the 3D representation of graph embedding captured by **graph sage** network on **en-fr** paired data with help of **t-sne.**

2.8 OBSERVATION:

- The number of trainable parameters is defined for each model in section 2.6. The highest parameter is for Graph Sage and the minimum for CNN

- Every deep network acts differently on different datasets there isn't any consistency in performance across the dataset. For example en-zh and zh-en are the same graphs only the train and test set are a little different though we get varied performance over the different neural networks.

- CNN outperforms the other deep models for all the datasets followed by graph sage. The possible reason for CNN to achieve the highest accuracy may be the structured data

- The MLP model faced the highest overfitting since the beginning of the training.

- The data was small hence almost all the models were overfitting, so we had to tune the hyperparameters to make the validation and training accuracy converge together.

- The embeddings in the t-sne diagram show that the data points belonging to different languages have clustered together.

2.9 CONTRIBUTION:

The work for the project was equally divided among the two of us. We equally contributed to each section of the project coding and writing. Data processing, MLP, GCN was done and tuned by Mriganka and CNN , Gsage, GAT was done and tuned by Ananya. The model architecture and the training have 50-50 contributions.