International Journal of Scientific Research and Management (IJSRM)
||Volume||10||Issue||03||Pages||EC-2022-772-782||2022||
Website: www.ijsrm.in ISSN (e): 2321-3418
DOI: 10.18535/ijsrm/v10i3.ec04

How to optimize Recommendation System Performance using Deep Neural Network based Graph Architecture

[1]Indranil Dutta

[1] Data Science and Advanced Analytics, Bangalore/560037, India.

Abstract
In this research, would bring a description and comparison of how the Deep learning-based Graph neural Network actually outperforms the other similar recommender system like collaborative filtering, content-based filtering, SVD, Matrix Factorization and few others. This is basically achieved by exposing the correct relation between the objects through a graph architecture and the dependency and inter correlation between them. Would also like to share an in-depth analysis and understanding of how the Graph architecture works and the underlying theories. This could be either a TensorFlow based architecture or a Pytorch based architecture but in this paper will mainly focus on the TensorFlow one for its flexibility and cloud friendly nature for adopting in any framework.

Key Words: Recommendation System, TensorFlow, Graph Architecture, Neural Network

1. Introduction

We live in the jungle of digital world where we are more connected than ever before. Not only is the world connected, but the data we collect is more and more related. Sometimes we are storing the data in a tabled format for better accessibility in a normalized mode, but the underlying connectivity and relations have been neglected at a significant extent. Here's some the concept of graph architecture where learning useful representations from objects structured as graphs is useful for a variety of machine learning applications such as social and communication networks analysis, biomedicine studies, and recommendation systems. Graph representation learning aims to learn embeddings for the graph nodes, which can be used for a variety of ML tasks such as node label prediction (i.e., categorizing an article based on its citations) and link prediction (e.g., recommending an interest group to a user in a social network).

In today's landscape lot of players are now building the solution to come up with very accurate and scalable to meet the business requirement. But in order to bring the accuracy either the solution would be complicated and heavy which didn't satisfy the scaling criteria, otherwise people end up with a very naïve and fast solution which again fail to satisfy the end users accuracy. But by implementing the Graph Neural Network (GNN) a comparatively optimized solution comes in the picture which takes care of the accuracy at great extent as well the scalability. In practical scenario there is a huge chunk of data is generating every moment and is waiting for a meaningful recommendation. Simple collaborative or content-based approach finds difficulties to handle the volume of data thereby the scalability comprises at some extent. This is also another angle to encourage using a Neural Network architecture which can be deployed (Like TFX in GCP) in a very scalable framework to maintain the expectation

2. Graph Functional Context

A typical graph architecture consists of nodes and edges (their relationships). This simple yet very effective way of storing the data allows you to model almost any real-world scenario. A graph can represent Social Networking Analysis, Fleet Movement Managements, Customer purchase preferences, protein interactions, chemical bonds and many others. Now, the objective of creating the deep learning-based graph architecture is how we can bring the graph functionality along with the power of AIML to project some meaningful insights of the real-world problems. In recent years, a lot of research was made on extracting network structure that graph networks can support two critical capabilities:

- **Relational reasoning**: Explaining relationships between different entities, such as image pixels, words, proteins or others

- **Combinatorial Generalization**: Constructing new inferences, predictions, and behaviours from known building blocks.

A typical input for any machine learning model is a vector that represents each data point. The process of translating relationships and network structure into a set of vectors is named node embedding. A node embedding algorithm aims to encode each node into an embedding space while preserving the network structure information. Here's the task is to identify the correct context of each objects to derive a more meaningful vector representation. There is a gamut of open-source vector embedding solutions are available which is a brad of butter of a typical NLP project but in graph architecture this has extend a new dimension of using the similar techniques to gather a meaningful input before feeding the same in any complex model. One of the main embedding concepts that will be highlighting in this paper is node2vec.

3. Underlying Concept Behind node2vec

Information flows in different directions through a graph architecture. Here is an example of a linear unidirectional structure to explain how the objects are connected with each other and the blue arrows defines their relationship and at what extent one is dependable on other.

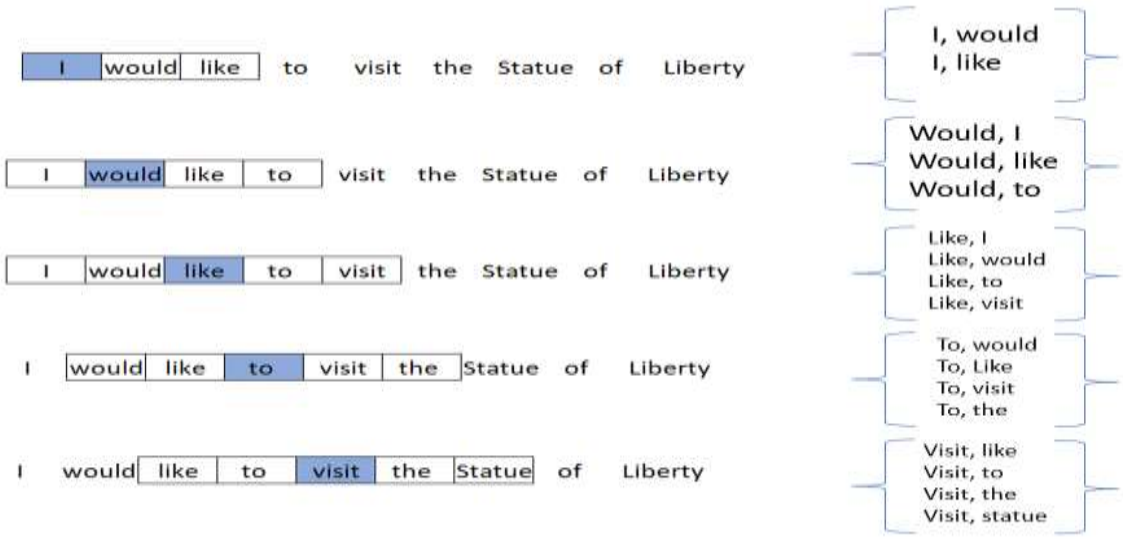
Figure -1: A typical graph flow with nodes and edges

In this context the node2vec algorithm is mainly inspired by the word2vec architecture. Word2vec also followed through two algorithms CBOW and skip-gram model. The node2vec is mainly derive the concepts from skip gram model. Therefore, to properly understand node2vec, need to first discuss how the word2vec algorithm works.

word2vec is a shallow, two-layer neural network that is trained to reconstruct linguistic contexts of words. The objective of the word2vec model is to produce word representation from a text corpus. Word representations are positioned in the embedding space such that words that share common contexts in the text corpus are located close to one another in the embedding space.

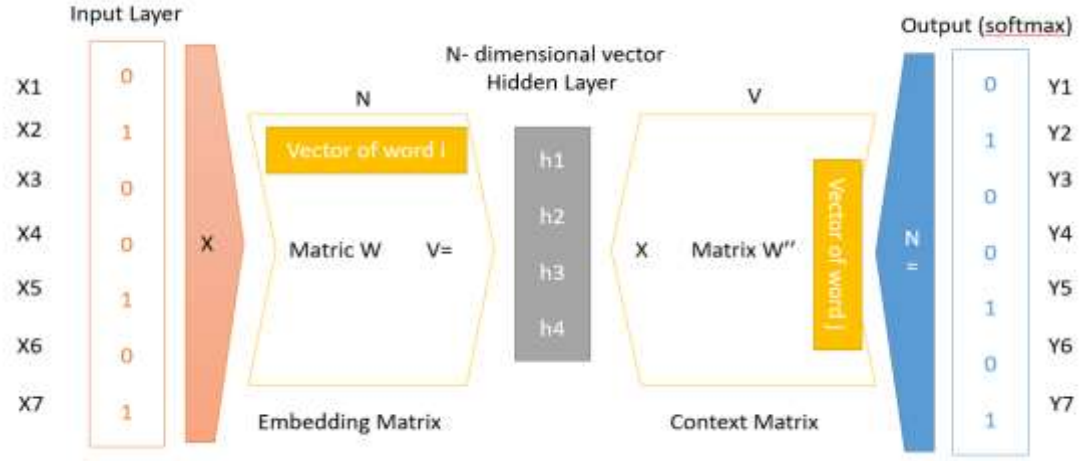
As node2vec is inspired by the skip-gram model, so we will not elaborate the CBOW implementation explanation. The skip gram model predicts the context for a given word. The context is defined as the adjacent words to the input term.

Figure -2: A skip gram model explanation



The word highlighted in blue is the input term. In this example, the context window size is varying from three to five. Window size is defined as the maximum distance between words in the context window with the input word in the center. Here in the first line, we are concerned about the common content of the words I and would, I and Like as I is the input words. If we see the last example where we maintain the window size is 5, the context of the word visit would be more clearly identified by sharing the common context of other adjacent words. This is not only converting the text in just a numeric matrix but also adds a proper context of where the words is using and what could be the potential meaning of the words in that context.

Figure -3: A skip gram model architectural diagram



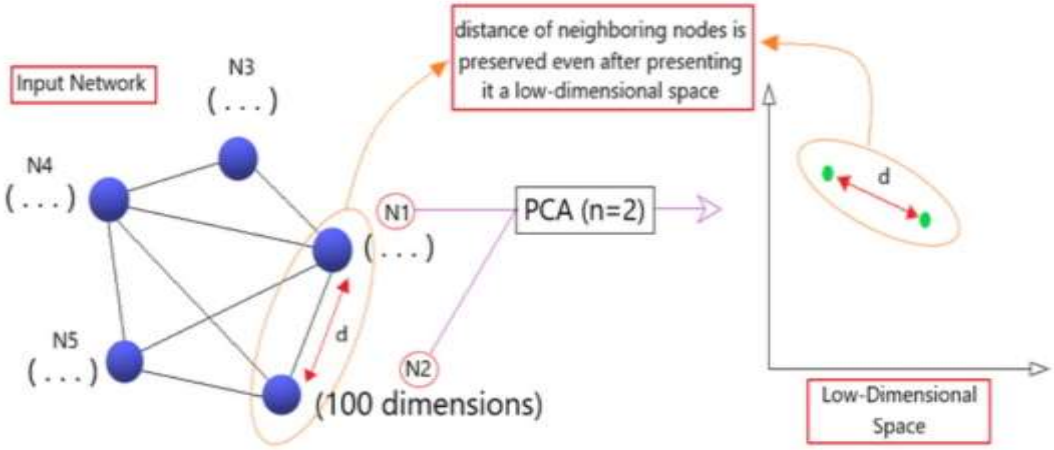
The architecture of the skip-gram model is basically showing while training this neural network, the input is a one-hot encoded vector representing the input word, and the output is also a one-hot encoded vector representing the context word. We have already discussed how the skip gram model combines words to discover the common context based on the input words and the window size. word2vec mostly concerned about the fact that we aren't interested in the output vector of the neural network, but rather concerned on how to learn the weights of the hidden layer. The weights of the hidden layer are actually the word embedding we are trying to learn. The number of neurons in the hidden layer will determine the embedding dimension or the size of the vector representing each word in the vocabulary.

Similarly, node2vec is based on the principle of learning continuous feature representation for nodes in the graph and preserving the knowledge gained from the neighbouring N nodes. Assume there is a graph having

a few interconnected nodes creating a network. The node2vec algorithm learns a dense representation (say 100 dimensions per feature) of every node in the network. If these 100 dimensions of each node are plotted in a 2D graph by using dimensionality reduction techniques, the distance between 2 nodes in the low-dimensional graph is the same as their actual distance in the given network.

In this way, the framework maximizes the likelihood of preserving neighbouring nodes even if you represent them in a low-dimensional space.

Figure -4: Dimension reduction in node2vec



Negative sampling and subsampling

There are two optimization techniques to fine tune the concept behind nove2vec.

The first optimization is the **subsampling of frequent words**. In any text corpus, the most frequent terms will most likely be the so-called stop words. But the skip-gram model really gains benefit from co-occurrences of words like “Fruit” and “Mango”, the benefit of frequent co-occurrences of “Fruit” and “the” (one of the common stop words) is far smaller as nearly every word is accompanied with “the” term. Here is the context of subsampling technique to address this issue. For each word in the training corpus, there is a chance that we will ignore specific instances of the word in the text. The probability of omitting the word from a sentence is related to the word’s frequency. The subsampling of frequent words during training results in a significant speedup and improves the representations of less frequent words.

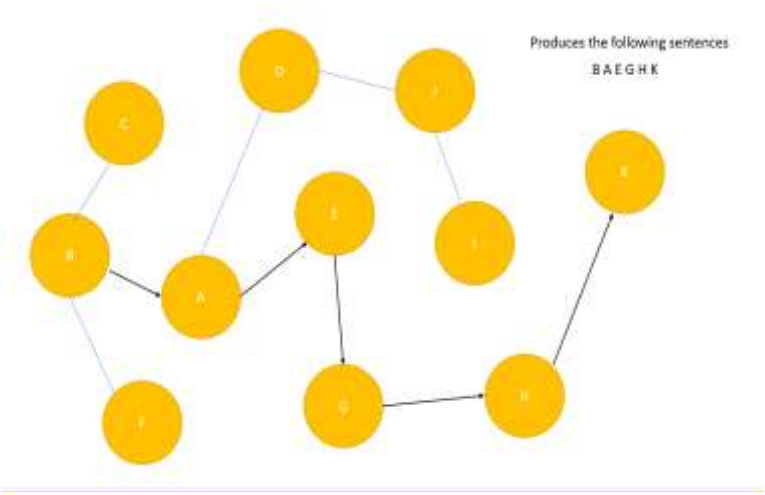
The second optimization is the introduction of **negative sampling**. Training a neural network involves taking a training sample and adjusting all the neuron weights to predict that training sample more precisely. Now, each training sample will adjust all the weights in the neural network. Updating several thousand weights for every input-context training pair is very computational heavy and occupies lot of resources. Negative sampling solves this performance issue by having each training sample modify only a small subset of the weights rather than all of them. With negative sampling, a “negative” word is one for which we want the neural network to output a 0, meaning that it is not in the context of our input term. We will still update the weights for our “positive” context word. In a unigram distribution, more frequent words are more probable to be selected as negative samples (like stop words).

Node2vec

The node2vec algorithm uses skip-gram algorithm with negative sampling. Now the algorithm and its optimization techniques are created but the next task is to create the text corpus for the model. The node2vec uses random walks to generate a corpus of “sentences” from a given network.

A random walk can be interpreted as a drunk person traversing the graph. We cannot anticipate the next step of such move but definitely it will move to some of the nodes. A drunk person traversing the graph can only jump onto a neighbouring node.

Figure -5: Random walk theory

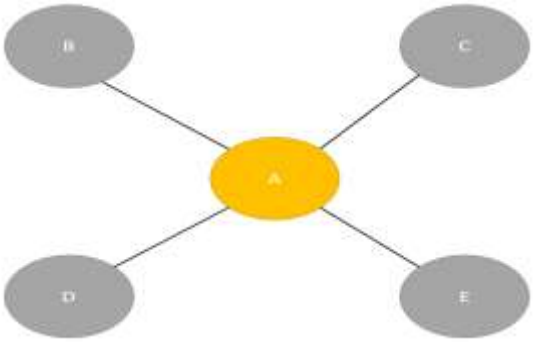


Assume the drunken person start traversing the graph from node B. At random, you pick a neighbouring node and jump on to it. Then the person repeats the same process until a predefined walk length. The walk length parameter defines how long the "sentences" will be. For every node in the graph, the node2vec algorithm generates a series of random walks with the particular node as the starting node. We can define how many random walks should start from a particular node with the walks per node parameter. Here we can take an example of B-A-E-G-H-K as one of the random paths, similarly we can anticipate another path like B-A-D-J-I. Now, to sum it up, the node2vec algorithm uses random walks to generate a number of sentences starting from each node in the graph. The walk length parameter controls the length of the sentence. Once the sentences are generated using random walks, the algorithm inputs them into the skip gram model and retrieve the hidden layer weights as node embeddings. That is the whole theory of the node2vec algorithm.

However, the node2vec algorithm implements second order biased random walks. Now let's discuss on how the first order and second order random walk works practically.

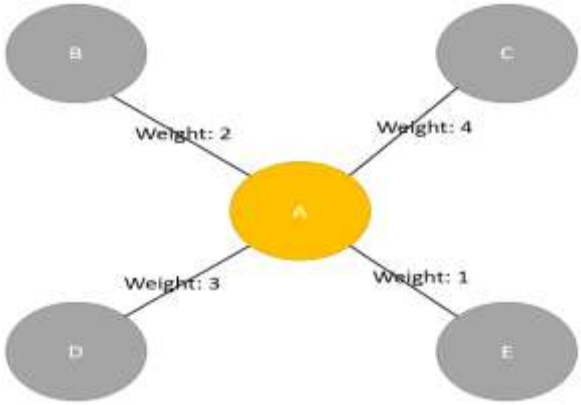
First-order random walks

Figure -6: First order random walk



Let's assume a situation where we have somehow stand up at node A. Now the first-order random walk only concerned about at its current state, the algorithm doesn't know which node it was at the earlier step. Therefore, the probability of returning to a previous node or any other node is equal. There is no advanced math concept behind the calculation of probability. Node A has four neighbours, so the chance of traversing to any of them is 25%. Now in case the graph is weighted, meaning that each relationship has a property that stores its weight. In that case, those weights will be included in the calculation of the traversal probability.

Figure -7: First order random walk with weight

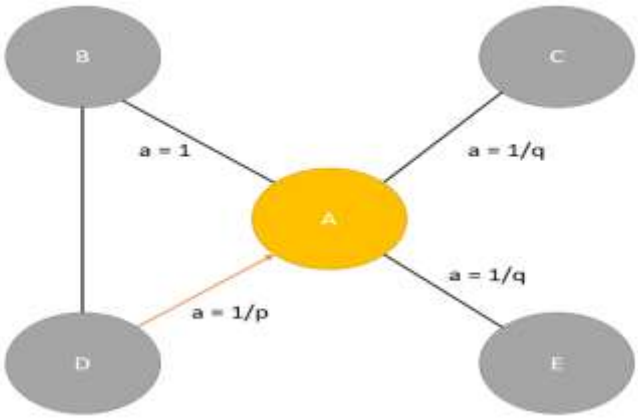


In a weighted graph, the chance of traversing a particular connection is its weight divided by the sum of all neighboring weights. For example, the probability to traverse from node A to node E is 1 divided by 10 (10%) and the probability to traverse from node A to node D is 3 divided by 10 i.e., 30%.

Second-order biased random walks

In case of Second-order walks, the algorithm takes into account both the current as well as the previous state. To put it simply, when the algorithm calculates the traversal probabilities, it also considers where it was at the previous step.

Figure -8: Second order random walk



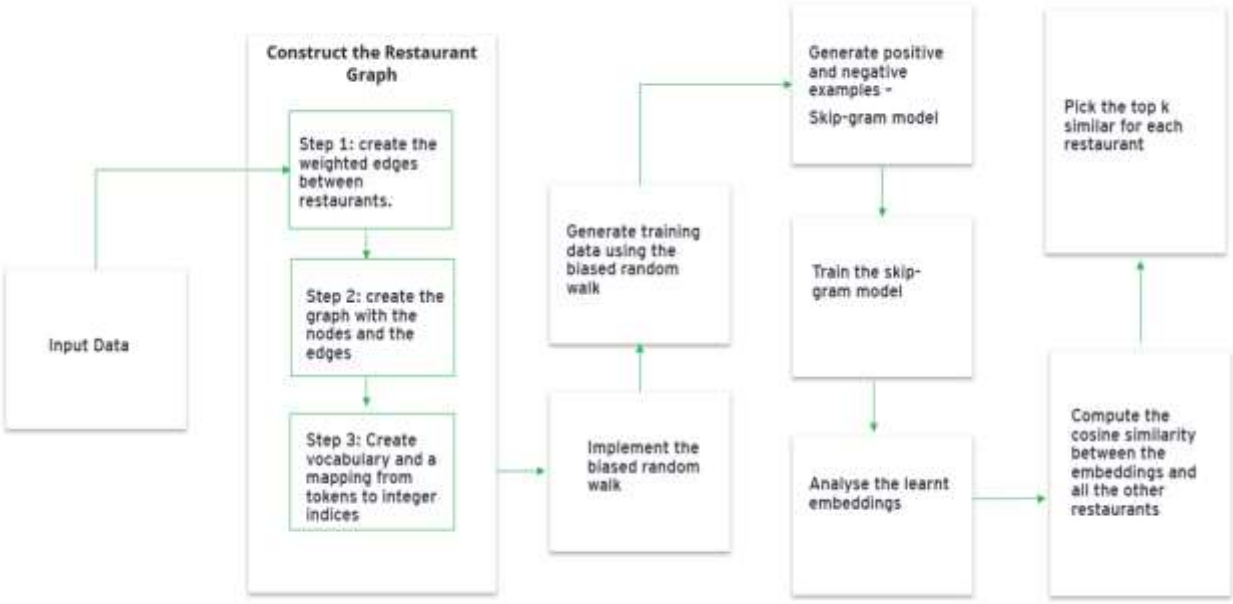
The walk just traversed from node B to node D in the previous step and is now evaluating its next move. The likelihood of backtracking the walk and immediately revisiting a node in the walk is controlled by the return parameter p. Setting a high value to parameter p ensures lower chances of revisiting a node and avoids 2-hop redundancy in sampling. On the other hand, if the value of the p parameter is low, the chances of backtracking in the walk are higher, keeping the random walk closer to the starting node.

The In and Out Parameter

The in-Out parameter q allows the traversal calculation to differentiate between inward and outward nodes. Setting a high value to parameter q ($q > 1$) biases the random walk to move towards nodes close to the node in the previous step. Going to the previous image, if you set a high value for parameter q, the random walk from node D is biased towards nodes closer to node A. Such walks obtain a local view of the underlying graph with respect to the starting node in the walk and approximate **breadth-first search**. In contrast, if the value of q is low ($q < 1$), the walk is more inclined to visit nodes further away from node D. This strategy encourages outward exploration and approximates **depth-first search.**

4. Solution Design

Figure -9: A generic flow of a GNN solution



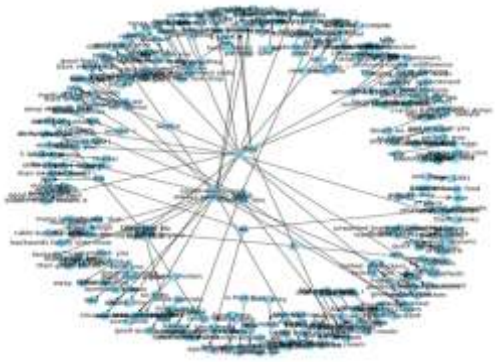
To explain the following concept let's take an example of an open-source movie lens dataset.

Step 1: create the weighted edges between movies.
First, we have to create an edge between two nodes in the graph if both movies are rated by the same user >= min_rating. The weight of the edge will be based on the **pointwise mutual information** between the two movies, which is computed as: log(xy) - log(x) - log(y) + log(D), where:

- xy is how many users rated both movie x and movie y with >= min_rating.
- x is how many users rated movie x >= min_rating.
- y is how many users rated movie y >= min_rating.
- D total number of movie ratings >= min_rating.

Pointwise Mutual Information
The pointwise mutual information represents a quantified measure for how much more- or less likely we are to see the two events co-occur, given their individual probabilities, and relative to the case where the two are completely independent.

Figure -10: A typical GNN graph structure



Step 2: create the graph with the nodes and the edges
To reduce the number of edges between nodes, we only add an edge between movies if the weight of the edge is greater than min_weight.

Step 3: Create vocabulary and a mapping from tokens to integer indices
Implement the biased random walk
A random walk starts from a given node, and randomly picks a neighbour node to move to. If the edges are weighted, the neighbour is selected probabilistically with respect to weights of the edges between the current node and its neighbours. This procedure is repeated for num_steps to generate a sequence of related nodes.

The biased random walk balances between breadth-first sampling (where only local neighbours are visited) and depth-first sampling (where distant neighbours are visited) by introducing the following two parameters:

1. Return parameter (p): Controls the likelihood of immediately revisiting a node in the walk. Setting it to a high value encourages moderate exploration, while setting it to a low value would keep the walk local.
2. In-out parameter (q): Allows the search to differentiate between inward and outward nodes. Setting it to a high value biases the random walk towards local nodes, while setting it to a low value biases the walk to visit nodes which are further away.

Step 4: Generate training data using the biased random walk

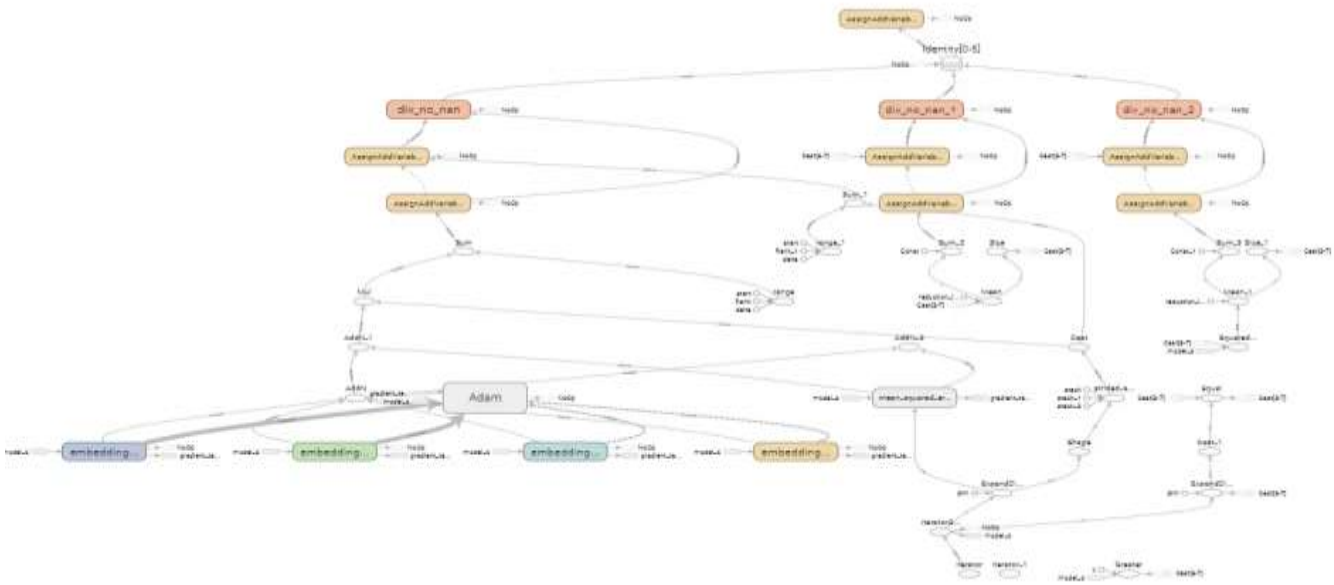
Generate positive and negative examples
To train a skip-gram model, we use the generated walks to create positive and negative training examples. Each example includes the following features:

1. target: A movie in a walk sequence.
2. context: Another movie in a walk sequence.
3. weight: How many times these two movies occurred in walk sequences.
4. label: The label is 1 if these two movies are samples from the walk sequences, otherwise (i.e., if randomly sampled) the label is 0.

Step 5: Train the skip-gram model
Our skip-gram is a simple binary classification model that works as follows:

1. An embedding is looked up for the target movie.
2. An embedding is looked up for the context movie.
3. The dot product is computed between these two embeddings.
4. The result (after a sigmoid activation) is compared to the label.
5. A binary cross entropy loss is used.

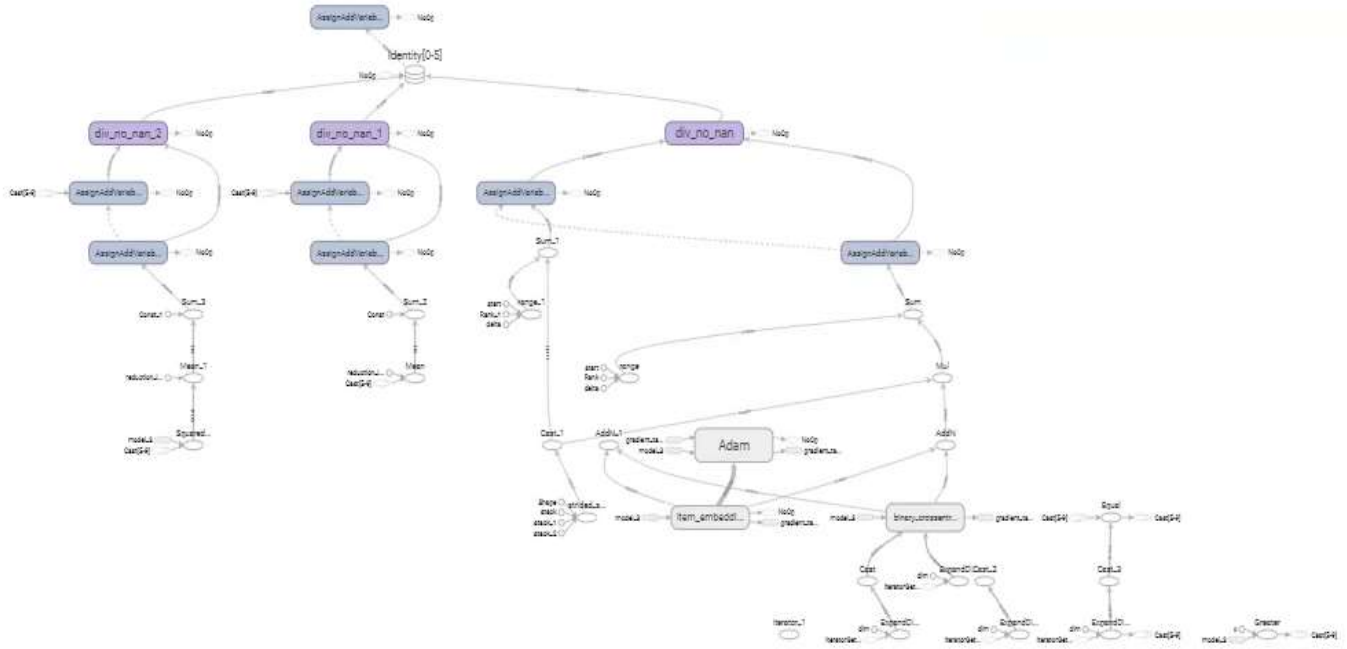
Figure -11: A GNN Tensor board architecture



After training the model we can see the GNN structure from the tensor board graph option. Based upon the size of data and no of nodes and edges the structure would be complex.

Side by side we have also trained a Naïve Neural Network – Multi layer perceptron model with the same dataset and here is the tensor board representation of the MLP architecture which is comparatively simple than the GNN one but in terms of the accuracy and error there is a significant contrast.

Figure -12: A MLP Tensor board architecture



Step 6: Result comparison and freeze the approach

We have considered the RMSE to define the model efficiency and accuracy and from the below mentioned chart its very clear all the base models like collaborative, content-based models or SVD are not brings comparative result that the neural network-based architecture draws. And finally, out of them the loss has drastically minimized by the GNN and we conclude that the GNN is the optimized solution for this type of recommendation solution.

SL No	Algorithm Name	RMSE
1	Collaborative Filtering (Item - Item)	1.58
2	Knearest Neightbour	1.85
3	SVD	1.93
4	SVD++	2.02
5	Slope- One	1.45
6	C0-Clustering	1.34
7	Multi Layer Perceptron	0.98
8	**Node2vec (Graph Neural Network)**	**0.05**

5. Sample Use Case Fits Inthis Architecture

There are few sample traditional AIML use case that fits in this stricture and performs well:

- E-commerce recommendation (Amazon/Flipkart)
- Netflix/amazon prime recommendation
- Chemical component bonding (Lupin/Dr Reddy)
- Social Networking Analysis (Facebook/Twitter)
- Fleet management services (Uber/Gojek)
- Product Merchandizing (H&M/ M&S)
- Logistics and SCM (Shell/ Fed-Ex)

6. Benefits

- Easily identify the hidden relationship at best capability.
- Accurate recommendation (limited bias)
- High scalability and adoptability
- Framework agnostic
- Diversified application and acceptance
- Scalable and portable
- Easily trained on CPU/GPU or TPU
- Effective on very large volume of data.

7. Limitations

There are few limitations as well to use the architecture:

- Complicated architecture hence maintenance is not easy.
- Always require high volume of data to get the best possible outcome
- Computational heavy, those scenarios where we have less volume of data and limited infrastructure, this solution wont suits properly.

8. Conclusions

The solution is comparatively new and complicated and absorbed by limited enterprises for their business solution, but because of its promising results and robust architecture it has immense demand among data scientists. In this paper I have tried to bring utmost all required concepts related to the GNN to give a brief understanding to the readers as well as a practical implementation with real tensor board architectures and model results comparison to help reads understand the code idea of the entire analysis. Like all other

solutions it also has some pros and cons but in a nutshell any enterprise or business has the capability to provide standard infrastructures and dealing with a large volume of data, then this type of solution definitely outperforms other naïve approaches at a very good extent. My humble request to all of the reader to first go through the core components of the paper and follow the below mentioned reference links to understand the in-depth concepts before directly jump to the implementation part.

Reference

1. Graph neural networks: A review of methods and applications (arxiv.org)
2. A Gentle Introduction to Graph Neural Networks (distill.pub)
3. https://arxiv.org/pdf/1901.00596.pdf
4. Graph Neural Network and Some of GNN Applications: Everything You Need to Know - neptune.ai
5. Graph Neural Networks: A learning journey since 2008— Part 1 | by Stefano Bosisio | Towards Data Science

BIOGRAPHIES

Indranil Dutta is a Principal consultant and Lead Data scientist with more than 11 years of rich experience in Data science and Artificial Intelligence in various industries. His core competency is in delivering scalable Data science and AI projects in Big data and Cloud environments.