Evaluating Vertex Embeddings using Shortest Path Distance Approximation

Andor Diera
Ulm University
andor.diera@uni-ulm.de

Karsten Kögel
Ulm University
karsten.koegel@uni-ulm.de

Marios Sirtmatsis
Ulm University
marios.sirtmatsis@uni-ulm.de

Tushar Singhal
Ulm University
tushar.singhal@uni-ulm.de

ABSTRACT

Dijkstra, BFS, A*, Bellman Ford are amongst the first algorithms that come to mind when we try to calculate the Shortest Path Distances between two vertices of a graph. Though these classical algorithms represent the cornerstone of many graph related tasks, they do not scale well on today's massive networks. In this paper we explore in detail the method proposed by Rizi et al.(2018) to approximate the shortest path distance between a pair of vertices using deep neural networks. We compare different vertex embeddings along with different landmark selection techniques on this downstream task and present a new convolution operation to approximate any binary operation for creating joint input representations. More notably we also propose a novel approach to evaluate the accuracy of structural vertex embeddings, through the use of the Shortest Path Distance Approximation task.

1 INTRODUCTION

Conventional computer algorithms are immensely adept at solving graph related task, but they must be programmed manually and in many cases they don't scale well on today's enormous graph-based databases, such as modern social networks, traffic networks or protein-protein interaction networks. Neural Networks(NNs) on the other hand, serve as a black-box approach to solving a multitude of tasks, by representing any arbitrary continuous function given appropriate weights [21]. One of the great advantages of NNs is that once they are trained, the time complexity of any new inference remains constant($O(c)$)[33]

Graph Neural Networks (GNNs) combine the expressive power of graph-based input data with deep learning based methods. Although there have been a few studies done on combining feed-forward and recurrent neural networks with graphs [13, 37] in the nineties, GNNs remained somewhat obscure until the recent advancements in computer vision with Convolutional Neural Networks (CNNs)[25] and representation learning with word embeddings in Natural Language Processing(NLP)[29]. Using techniques found in CNNs and/or in representation learning, variants of GNNs are able to efficiently work with graph data and model elements of the graph and their dependencies[45].

In our study we focus on graph representation learning, or more specifically on learning vertex representations and using them as an input to approximate the shortest path distance between two vertices, using supervised learning. While direct embedding methods lack the ability to generalize (meaning they can't be used on dynamic graphs or be reused for new graphs), they are more flexible and cost effective than traditional machine learning methods in graph analysis. We base our research on the work done by Rizi et al [33]. In their work they used the node2vec[16] and the Poincare [30] algorithms to learn vertex embeddings which were then combined into pairs of two using different binary operations and fed in to a simple regression-based Deep Neural Network(DNN). With this simple approach they managed to achieve great accuracy on shorter path lengths, but did not succeed in approximating longer path lengths correctly. We hypothesize that for reaching better performance on longer paths, the vertex representations should capture sufficient information of high-order proximities and during training data creation the landmark vertices should be chosen carefully.

To evaluate this assumption in detail we compare different vertex embedding techniques and landmark selection approaches on two social network datasets, using the Shortest Path Distance Approximation(SPDA) as a downstream task. We also present a new approach to replace the binary operators with a simple convolutional layer during vertex representation concatenation, and compare this approach to the best binary operators found in the paper by Rizi et al.[33]. Finally we also propose that the task of SPDA could serve as a precise evaluation method for structural vertex embeddings.

2 RELATED WORK

2.1 Vertex Embeddings

The idea behind creating vertex embeddings of a given graph is to encode global and local structural information of a vertex in a way, where for each vertex $v_i \in V$ in the graph G, a vector with dimension d is created. Vertices that are similar, measured through some similarity function in the graph should be close to one another in the embedding space and far away if dissimilar. Over the past years, plenty of approaches and techniques have been proposed to solve the task of embedding vertices.

The first class of approaches to generate vertex embeddings consists of methods where embeddings are generated by using shallow NN architectures. DeepWalk [32] is one of the most well-known algorithm using this approach and in general one of the first to utilize NNs in creating vertex embeddings. It leverages advancements in the field of NLP, more specifically the SkipGram model [29], to generate representation vectors for all vertices $v_i \in V$ with $i = 0, ..., |V|-1$. It does this by generating random walks with length t and passing them to the SkipGram model, where they are being treated as sentences. Intuitively, DeepWalk optimizes embedding

vectors $z_i, z_j \in Z$, corresponding to vertices $v_i, v_j \in V, i \neq j$, to be close to one another if the probability of a random walk starting at v_j and visiting v_i, $p_{G,t}(v_i|v_j)$, is high. Improving upon this approach, node2vec[16] discards the use of completely random walk sequences and includes two parameters to bias random walks for creating a trade-off between global graph structure exploration and local community structure observation. This novel improvement is more suitable for task specific random walk generation. For being computationally feasible, DeepWalk uses a hierarchical softmax to approximate $p_{G,t}(v_i|v_j)$ while node2vec uses negative sampling.
Different to these random-walk based approaches, LINE uses graph-tailored objective functions to both capture first and second order proximity between vertex pairs, embedded in the latent space [38]. First order proximity represents the direct connection between vertices while second order proximity measures the similarity of two vertices' neighbourhoods. However, LINE's objective functions either capture first or second order proximity and generate embeddings independently, which are then concatenated to capture both orders of proximity. As a member of the second class of embedding approaches, DNN based techniques, such as SDNE applies a similar idea as LINE but instead uses a deep auto-encoder architecture to jointly optimize one objective function to create embeddings capturing both, first and second order proximity [42]. Additional embedding techniques using deep learning methods are DNGR and GCN [7, 23]. Similar to SDNE, DNGR uses an auto-encoder network to generate embeddings. GCNs define a message passing operation on graphs to aggregate neighbourhood information of vertices. They do this by iteratively updating the representation of a vertex, the aggregation of its neighbors.

There also exist factorization based methods, for embedding vertices into a low-dimensional latent space, including Locally Linear Embedding (LLE) [34], Laplacian Eigenmaps [3], Cauchy Graph Embedding [27], Structure Preserving Embedding (SPE) [36], Graph Factorization (GF) [2], GraRep [6], HOPE [31] and other dimensionality reduction techniques like Principal Component Analysis (PCA) [22], Linear Discrimant Analysis (LDA) [28], ISOMAP [40], Multidimensional Scaling (MDS) [10], Locality Preserving Properties (LLP) [20] and more. All of these methods represent graphs as matrices in some way and most of them use matrix factorization techniques to generate low-dimensional embeddings. This, however, comes with a high cost in time complexity.

2.2 Shortest Path Problem

The Shortest Path Problem(SPP) is one the most renowned algorithmic task in graph theory. It deals with finding the shortest weighted path between two vertices of a graph. It is a task that has many real world applications, most notably in navigation or in calculating degrees of separation in social networks. Not surprisingly SPP has a long history in theoretical computer science, and many algorithms have been developed to solve this task.
The most famous algorithm for calculating the shortest path is the Dijkstra algorithm[11] which was developed in the 1950s. The worst-case time complexity for the generalized Dijkstra algorithm is $O((|E| + |V|) + log|V|)$. A big disadvantage of this algorithm is that it only works on non-negative edge weights. For solving the SPP on graphs that contain negative weights as well, the Bellman-Ford algorithm[4, 12] can be used. Although it is more versatile than Dijkstra, it is slower with a worst-case time complexity of $O(|V| * |E|)$.
A more advanced version of Dijkstra is the A* algorithm[18], which combines heuristic approaches such as Greedy Best-First Search with Dijkstra to execute an informed search. While in practice A* works very well, and is still the most used SPP algorithms in many applications (for example pathfinding in video games), it possesses the same worst case time complexity as Dijkstra.
In the case of simple unweighted graphs, the classic Breadth-First Search algorithm can be also used which has a worst case time complexity of $O(|V| + |E|)$.

2.3 Link Prediction

Link Prediction is a common binary classification task in the graph domain, that predicts whether or not an edge exists between two vertices[35][14][26][39][19]. This problem can be also seen as a bounded Shortest Path Distance Approximation task, where a NN predicts whether two given vertices have a 1-hop distance between them. The task of Link Prediction is typically approached by creating a compact representation of vertices of a graph such as vertex embeddings which contain both vertex features and topological features. Then pairs of vertices are sampled from the network to create a training and testing data set. Firstly vertex pairs with no edge between them are selected that are labelled as negative samples, and then vertex pairs connected by an edge are selected and labelled as positive samples.
To get a compact representation of the input vertices, algorithms such as node2Vec are used. Since the output obtained from these algorithms give a vector representation, there are a variety of ways in which they could be passed to the neural network as inputs. Generally a binary operation is performed on two vertex embeddings to get a singular representation for the edge between them. These binary operations contain basic algebraic operations such as addition, subtraction, multiplication or averaging. In some cases both vertex embeddings are passed to a multi-head neural network, in other cases the vector representations of the vertices are simply concatenated. These latter techniques are usually less reliable and give worse performance than most of the binary operators.
The perfect embedding for link prediction should have high proximity scores between a vertex and all the vertices that are at one hop distance from it. And it should have a low to zero proximity score between any 2 non-connected vertex pairs. These scores are very hard to maintain, and this difficulty increases with the complexity of the graph structure. The clearer this boundary between the proximity scores is, the better is the NN in differentiating between connected and non-connected vertex pairs, and thus the task of link prediction. For this reason, it is more important to use efficient and fitting vertex representation than to increase the complexity of the NN architecture.

2.4 Shortest Path Distance Approximation

Rizi et al.[33] used deep learning to approximate the shortest path distance between two vertices in unweighted graphs. Similar to the task of Link Prediction they first created vertex embeddings

and combined them to a single representation with different binary operations. These input representation were then fed in to a simple regression-based DNN using MSE as a loss function. To create the training data set for supervised learning, the authors used the landmark approach[41] for precomputing the Shortest Path Distance(SPD) between a given number of selected vertices and all other vertices. The training set was built up from these vertex pairs and the calculated distance between them.
While their results indicate that their approach is only reliable for shorter path lengths (from length 5 the loss became significantly higher), the methods they presented leave a lot of space for future improvements. Although it is highly unlikely that using NNs to approximate SPD will ever get 100% accurate results, in some cases the low time complexity of new inferences ($O(c)$) can be more important than the total accuracy provided by the classical algorithms.

3 EXPERIMENTAL APPARATUS

3.1 Datasets

We worked with 2 data sets downloaded from a public repository [44] To evaluate our research hypotheses. Both data sets contain undirected and unweighted graphs, gathered from social networks.

- `Facebook`: This data set is compiled from Facebook-profiles connected via friendships. The vertices represent the profiles, while the edges represent the friendships between them.
- `Douban`: Douban.com, is a Chinese Web 2.0 website providing user review and recommendation services for movies, books, and music. The vertices represent users, while the edges represent the connections between them.

The properties of the data sets are summarized in Table 1 We selected two data sets that have large differences in size and structure, to test how our methods perform in different conditions. Most of our prototyping was done on the Facebook data set.

Table 1: Statistics of the Datasets

Dataset	Nodes	Edges
Facebook	4,039	88,234
Douban	154,907	327,094

3.2 Landmark Selection

We used the landmark approach to create the training and test sets for our DNN. The landmarks compromise a small subset of the whole graph. In the case of the Facebook data set the number of landmarks was 100 for the training set and 11 for the test set. Since the Douban data set was considerably larger the number of landmarks was restricted to 3 in the training data, and to 1 in the test set. Once the landmark vertices were chosen we calculated all shortest paths from these vertices to all other vertices of the graph using the Breadth-First Search algorithm. The vertex pairs are then linked up with the corresponding distance between them to create the ground truth for the network.
We theorized that the process of choosing these landmarks can have a large impact on the performance of the net, as indicated in [15]. To test this hypotheses we compared three ways to select landmarks in the Facebook data set, and used two of these approaches on the Douban graph.

- `random-based selection` : This approach works, as the name implies: by randomly selecting a given number of vertices as landmarks. While in theory this can result in sub-optimal edge-case selections, in practice it has been shown to work adequately.[33] [15]
- `community-based selection` : Our second approach was to search for communities in the graph, using the Clauset-Newman-Moore greedy modularity maximization[9] After the communities have been identified we selected a number of verties from them, according to their size compared to the whole graph.
  We this approach worked for the Facebook data-set where the number of identified communities was proportional to the size of the graph, we had to exclude this approach on the Douban data set, as it had too many communities 3.
- `coarsening-based selection` : In this approach we applied coarsening to the graphs using the coarsening part of the HARP algorithm.[8] Coarsening algorithms create a hierarchy of smaller (coarser) graphs with fewer vertices and edges, while still retaining the global structure of the original graph. We run the coarsening algorithm a until a coarser graph had equal or slightly more vertices then the number of landmarks, and sampled the given number of landmarks from this coarsened graph.

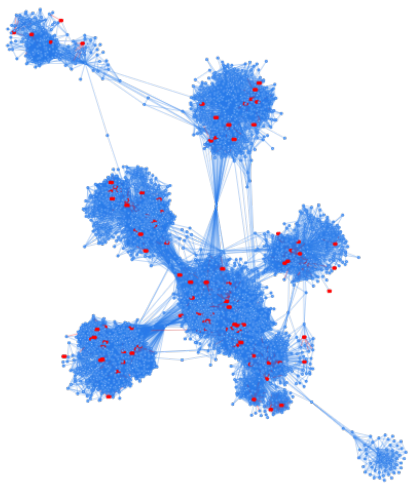

Figure 1: Landmarks(red) placed based on random selection in the Facebook dataset

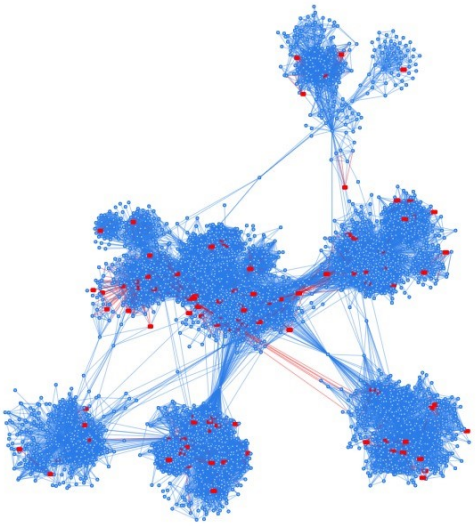

Figure 2: Landmarks(red) placed based on community selection in the Facebook dataset

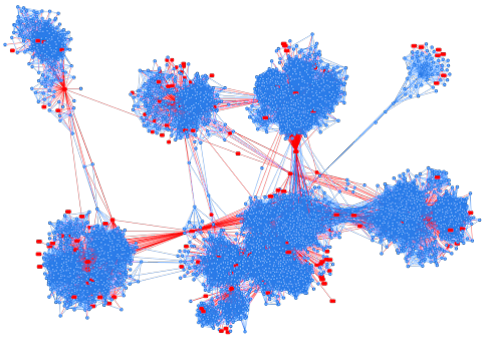

Figure 3: Landmarks(red) placed based on coarsening in the Facebook dataset

As it can be seen from visualizing these landmark selection approaches, in the random selection approach some smaller clusters in the graph can have zero landmarks, while both the community-based and coarsening-based selections are able to cover all parts of the graph.

We were not able to visualize the Douban dataset, due to its size, but according to the vertex to edge ratio, and the even distribution of shortest path lengths (compared to the Facebook data set which was heavily skewed to shorter lengths), we suspect that network was much less interconnected than the Facebook graph.

3.3 Vertex Embeddings

As a framework for generating embeddings, the GraphEmbedding library was used. We used 5 differenct vertex embedding methods in our final setup: node2vec, HARP node2vec, LINE, HARP LINE and SDNE. We also implemented a full GCN to compare its performance to our base approach with a simple regression network with vertex embeddings as inputs.

After optimization of the embeddings used in this work, following specifications showed to be best for the task of Shortest Path Distance Approximation. We set the *embedding dimension* for all embeddings to 128. For *node2vec* a *window size* of 10, a *walk number* per vertex of 10 with a *walk length* of 80 were used. More interestingly, figure 4 shows a diagram, visualising the optimization of parameters p and q. One can see, that even though there is no real pattern for when p and q bring best values for shortest path distance approximation, having both at 1 seemed to work best for the task. For *LINE*, we used 5 as a number of *negative samples*, a *batch size* of 512, *Adam* as optimization method with a *learning rate* of 0.1. Also, we trained the embeddings on both, first and second order proximity by concatenating first order proximity and second order proximity embeddings of size 64 for each vertex. We trained the *LINE* model for 50 epochs. For *SDNE*, we also used a *batch size* of 512, 10^{-6} for α and 5 for β. For the Neural Network Structure, similar to the initial *SDNE* implementation, we used $0.1 * |V|$ as the number of nodes in the first layer and 128 in the second layer. We trained the *SDNE* model for 40 epochs. For the embeddings created with HARP, we used the same configuration as described in the original paper.[8]

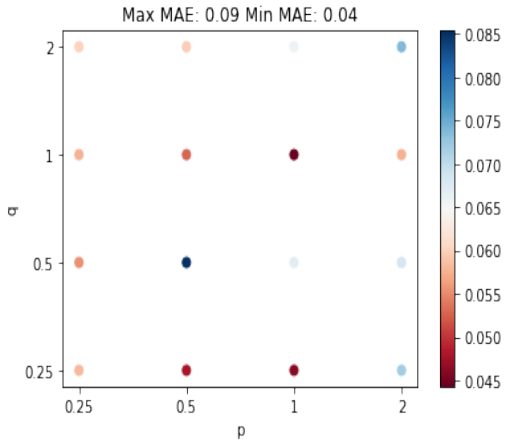

Figure 4: A visualisation of the optimization for the node2vec embedding. Each circle represents one configuration while the circle color indicates the configuration's MAE for the Shortest Path Distance Approximation task. One can see that there is no real pattern on when node2vec performs best with respect to p and q, but still p=q=1 appears to work best for the task of shortest path distance approximation.

3.4 Binary Operations

Given vertices v1 and v2, we need to extract their embeddings from the embedding matrix. The extracted vectors need to be passed to the neural network to make a prediction. These extracted vectors can be passed separately to a multi-head neural network or a joint representation can be generated. These joint representation can be created using binary element wise operations(averaging, subtraction,multiplication) on the 2 extracted vectors which do not increase the dimension of the initial vector, expect concatenation. From the results of the work done by Rizi et al.[33], we learnt that concatenation and averaging performed the best in most of the cases and subtraction was the worst, but these results varied between the data sets used in the study The authors did not present any theory why there was such a big variance in the performance between the binary operators on different data sets, and simply proposes a trial and error method to find the best fit.

Therefore we decided to fix the random seeds to get re produceable results. Upon running the experiments again we found out that average operation gave the best results. But in theory, one would expect that when using a concatenation operation, that the weights of the first layer would optimise through feedback and it would learn the average operation. Clearly this was not the case, so we came up with a novel idea of using the shared weights of a convolutional layer to estimate the best fitting binary operation.

Convolution weights can be seen as specific binary operations (0.5,0.5) for averaging, (1,-1) for subtraction etc. The weights are multiplied to each feature in the 2 vertex embeddings and then added to get a similar result as the binary operation. Since these weights are learned they can vary according to the back-propagation and adjust themselves with respect to the weights in the latter layers. Because of this they can achieve superior performance than averaging or any other fixed operation.

This works better because it introduces only 2 extra parameters by imposing 2 restrictions. Firstly, that feature at x position in the embedding vector of vertex v1 does not have any relationship with features in the embedding vector of vertex v2 expect at position x. So any binary operation result between those features is not calculated. Secondly, a similar binary operation is performed on each feature of the 2 vertex embeddings.

From our experiments we saw that the convolution weights are highly sensitive to weight initialisation, which has a major impact on the performance of the network. Whenever the weights were initialised with opposite signs it never achieved a good performance and using a print callback function we found that the best results were obtained when weights were in the range (1-1.5,1-1.5). Therefore we used a weight constraint while training to have both the weights above 0.

3.5 Regression Network

We used a simple DNN with 3 dense layers to solve the regression task. The layers have a ReLu a activation function between them followed by a Dropout Layer with a value of 0.4. The final layer is followed by a Softplus activation. A second network was used to implement the convolution operation on the input embeddings. It added a 2D Convolution after the input layer, which was followed by a Flatten layer before using the same architecture as described before. The base architecture can be seen in Figure 5.

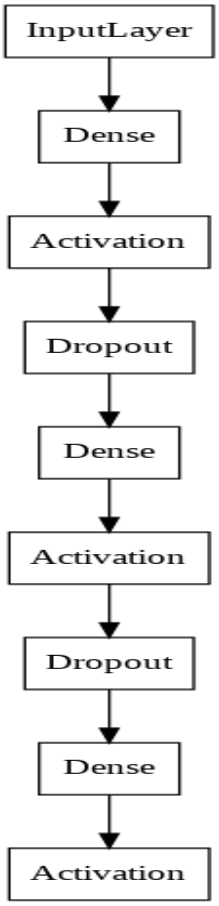

Figure 5: Regression network architecture

3.6 GCN

We used the GCN implementation provided by the Stellar Graph library. We chose to use a two-layer GCN, even though we needed a six-layer GCN to pass messages between 6 hop neighbours. The choice is based on the research outcome that a high number of layers in a GCN reportedly worsen the performance compared to a GCN with a low number of layers. We only use GCN for comparison with shallow embedding therefore, we do not use any of the state-of-the-art methods of link prediction like Residual Graph Convolution [5] or Variational Graph Auto-Encoders [24]. We trained the GCN for 5000 epochs since it utilizes a full batch, and there is no significant change in validation MSE after 5000 epochs. We use random feature vectors of size 128 which was shown to have the same performance as any other feature created with embedding methods.[1].

4 RESULTS

The table 2 show below contains the metrics from different datasets with the 2 binary operations and the convolutional operation across

	Facebook			Douban		
	Random	Community	Coarsening	Random	Community	Coarsening
node2vec						
avg	0.25 | 0.06 | 0.94	0.32 | 0.10 | 0.90	0.26 | 0.07 | 0.93	1.27 | 1.01 | 0.28	*	1.94 | 1.65 | 0.13
concat	0.33 | 0.11 | 0.89	0.35 | 0.12 | 0.88	0.38 | 0.15 | 0.86	1.16 | 0.90 | 0.30	*	3.36 | 3.24 | 0.00
conv	0.22 | 0.05 | 0.96	0.36 | 0.13 | 0.87	0.33 | 0.11 | 0.89	1.32 | 1.04 | 0.29	*	3.95 | 3.76 | 0.00
HARP node2vec						
avg	0.24 | 0.06 | 0.94	0.39 | 0.13 | 0.88	0.28 | 0.08 | 0.92	1.70 | 1.33 | 0.24	*	0.91 | 0.70 | 0.37
concat	0.29 | 0.08 | 0.92	0.40 | 0.16 | 0.84	0.36 | 0.12 | 0.88	1.47 | 1.11 | 0.31	*	1.32 | 1.05 | 0.28
conv	0.29 | 0.08 | 0.92	0.36 | 0.13 | 0.87	0.32 | 0.10 | 0.90	1.362 | 1.04 | 0.32	*	1.05 | 0.82 | 0.33
LINE						
avg	0.58 | 0.31 | 0.70	0.90 | 0.62 | 0.47	0.85 | 0.65 | 0.39	*	*	*
concat	0.67 | 0.40 | 0.62	1.26 | 0.92 | 0.32	1.91 | 1.34 | 0.28	*	*	*
conv	0.57 | 0.30 | 0.72	0.94 | 0.62 | 0.49	2.28 | 1.75 | 0.18	*	*	*
HARP LINE						
avg	0.46 | 0.19 | 0.82	0.62 | 0.36 | 0.65	0.51 | 0.24 | 0.77	1.73 | 1.38 | 0.22	*	0.98 | 0.75 | 0.35
concat	0.47 | 0.22 | 0.79	0.54 | 0.28 | 0.72	0.61 | 0.33 | 0.68	1.74 | 1.4 | 0.22	*	0.99 | 0.75 | 0.26
conv	0.45 | 0.18 | 0.83	0.55 | 0.26 | 0.76	0.68 | 0.42 | 0.59	1.37 | 1.09 | 0.27	*	1.09 | 0.85 | 0.30
SDNE						
avg	0.67 | 0.37 | 0.66	0.89 | 0.54 | 0.57	0.93 | 0.68 | 0.42	*	*	*
concat	0.77 | 0.46 | 0.60	0.95 | 0.66 | 0.46	0.98 | 0.72 | 0.40	*	*	*
conv	0.65 | 0.34 | 0.70	0.99 | 0.62 | 0.52	0.92 | 0.66 | 0.43	*	*	*
GCN						
avg	0.38 | 0.15 | -	0.38 | 0.16 | -	0.37 | 0.16 | -	*	*	*
concat	0.40 | 0.21 | -	0.41 | 0.21 | -	0.44 | 0.24 | -	*	*	*
conv	0.38 | 0.17 | -	0.36 | 0.20 | -	0.38 | 0.20 | -	*	*	*

Table 2: The result table including all assembled results from our experiments. Every entry is divided into three different evaluation metric measures. Entries are in the form *RMSE | MAE | ACCURACY*. The top columns indicate the dataset on which our methods were evaluated while the second-top columns indicate the used landmark selection technique, where *Random* is random selection, *Community* is community selection and *Coarsening* is coarsening. Metrics for entries annotated with a * were not feasible to be computed due to time or memory restrictions.

different embedding techniques. For most of the embeddings convolution performs better, still there is not enough evidence to support the claim that convolution performs significantly better than averaging and concatenation. Since RMSE and MAE only give a relative idea about the performance of the NN, we also use a accuracy score by rounding the regression output and comparing it to the ground truth. We got good results for the Facebook dataset but not so good results for the Douban dataset, even though it has a much more even distribution of the shortest path lengths. A possible reason behind that could be that the visualization of the Facebook dataset shows clear separation of the communities while we suspect that the Douban dataset does not, which makes it nearly impossible to find good landmark nodes. Hence the performance varies greatly with the new landmark selections every time.

5 DISCUSSION

As the main focus of this work was the comparison of several methods that can be used to approximate the shortest path distance using deep learning, the results need to be discussed in multiple aspects as well.
In the following part, the most important findings will be discussed in detail.

- `Convolution Operation`:
  We can infer from the results that concatenation did not perform better than any of the binary operators. One possible reason could be that the NN is unable to comprehend on its own the binary operators between all the possible feature combinations between the vertex pair embedding and in the

vertex embeddings.
The convolution operation performs a little better because it does not calculate operations on intra vertex features.

- `Random Landmarks`:
  Random selection of landmarks had overall the best results, with the best overall result being 96% Accuracy on the Facebook Dataset. This result was achieved in conjunction with the convolution operation and the node2vec embedding.
  This confirms the setup used by Rizie et al.[33], who only used random selection, but achieved comparable results to our study.

- `Community-based Landmarks`:
  This method produced the worst results with node2vec. These results were unexpected since this method is essentially the same as random, but with a guaranteed spread of landmarks in the communities. A possible explanation for these results is, the number of landmarks according to community size was wrong. In order to observe evenly spread landmarks, we selected more landmarks in larger communities. It could have been the case that an opposite selection would have worked since smaller communities may have needed more landmarks to represent them.

- `Coarsening-based Landmarks`:
  This was the fastest heuristic method we used, and had a similar performance compared to random selection in most of the cases, and in some cases it even outperformed it.
  This method has a similar aim as community-based selection, namely to assure a proportional representation of the landmarks, but instead of random sampling it focuses on choosing the truly important vertices. We believe coarsening methods are a good direction for choosing landmark vertices.

- `GCN`:
  GCN has a similar performance across all the landmark selection techniques. Looking at the binary operations we can see that averaging performed the best and concatenation the worst. Though the GCN is a much costlier method for this task than our simple DNN with vertex embeddings, we include it in the results to show that it is still better than some of the shallow embedding techniques for this shortest path distance task.

- `Vertex Embeddings`:
  Our initial hypothesis was that if a vertex embedding method performs well for the task of link prediction, then it should perform well for the task of shortest path prediction as well. But to our surprise Node2Vec outperformed all the embeddings which usually perform better in the link prediction. Both of these tasks rely on the assumption that vertices which have a similar representation in the feature space are close/connected in the graph as well.
  Link Prediction - For the task of link prediction the prefect vertex embeddings only have to maintain high proximity scores between adjacent vertex embeddings i.e. the 1 hop neighbours and low proximity scores for non connected vertices. It should also do not have a high similarity score for 1+i hop vertices.
  For our task of Shortest Path Distance Approximation, we need a embedding technique that not only maintains a high proximity score with 1 hop neighbours but also this similarity score should decrease as the distance between the 2 vertex increases.
  As stated in [17] SDNE should capture high order vertex features, thereby should be better suited for our task, but in our study it was under-performing.

5.1 A new evaluation method

There are a number of novel, promising embedding techniques which aim to capture local and global neighbourhood graph structural information. However, these techniques have only been used for the task of link prediction and node classification. We believe this is not a fair measure of an embedding strength since these tasks only answer binary questions like whether the two vertices are connected or whether or not two vertices belong to the same class.
During our experiments, we realized that the regression task would be a better way to compare how well vertex embedding techniques can represent the graph structural information. The shortest path distance approximation task can be extended to create baselines for comparing structural vertex embeddings.

We propose the following modifications:
1. We don't do a landmark selection instead we consider all possible pairs of vertices (connected and unconnected).
2. We create a dataset by calculating shortest path for connected vertex pairs and use 0 for unconnected vertex pairs.
3. Then we train a neural network with this dataset by using any of the binary operations.

Techniques to artificially balance the dataset could be also used to improve the NN performance. It poses great advantages as, in addition to approximating the connected and unconnected vertices, it can also approximate the distance between the connected vertices, modelled by the proximity score between the vertex pair embeddings.

6 CONCLUSIONS AND FUTURE WORK

In this work, we further investigated the task of Shortest Path Distance Approximation on multiple graphs, similar to Rizi et al. [33]. We evaluated different embeddings and configurations for the given task and introduced a new method to generate joint representation of these embeddings, based on a convolutional operation. With this, we were able to outperform Rizi et al. on the aforementioned Facebook graph. We also showed that complex architectures like GCNs do not necessarily result in better performance on tasks like the Shortest Path Distance Approximation. We also presented our idea to use the Shortest Path Distance Approximation task as a method to evaluate the magnitude of structural information perseverance of vertex embeddings. As future directions, it may be

interesting to once again explore other landmark selection techniques. It goes without saying that random landmark selection will always be the fastest compared to heuristic methods. Even though, random landmark selection may work quite well in practice, deterministic approaches are still more desirable theoretically. This is, because random behaviour can result in bad landmark selection, and therefore a bad representation of a graph inside the training set for a task, such as SPDA. This can particularly lead to bad results for the prediction of shortest path distances and therefore make the performance on this task unstable. Due to limited resources in time and computation, in this work, we were not able to evaluate our methods on multiple datasets and to scale to larger datasets. Therefore, investigating the performance of our approaches on larger graphs is of major interest and importance. Also, one could further dive into the use of specific GCN architectures for tackling SPDA. Another route could be to investigate on SPAGANs[43] for predicting shortest path distances between vertices, as it is more suited to preserving information about multiple neighbors of a vertex in a graph.

REFERENCES

[1] Ralph Abboud, Ismail Ilkan Ceylan, Martin Grohe, and Thomas Lukasiewicz. 2021. The Surprising Power of Graph Neural Networks with Random Node Initialization. https://openreview.net/forum?id=L7Irrt5sMQa

[2] Amr Ahmed, Nino Shervashidze, Shravan Narayanamurthy, Vanja Josifovski, and Alexander J Smola. 2013. Distributed large-scale natural graph factorization. In *Proceedings of the 22nd international conference on World Wide Web*. 37–48.

[3] Mikhail Belkin and Partha Niyogi. 2003. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation* 15, 6 (2003), 1373–1396.

[4] Richard Bellman. 1958. On a routing problem. *Quarterly of applied mathematics* 16, 1 (1958), 87–90.

[5] Xavier Bresson and Thomas Laurent. 2018. Residual Gated Graph ConvNets. arXiv:1711.07553 [cs.LG]

[6] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2015. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM international on conference on information and knowledge management*. 891–900.

[7] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2016. Deep neural networks for learning graph representations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 30.

[8] Haochen Chen, Bryan Perozzi, Yifan Hu, and Steven Skiena. 2018. Harp: Hierarchical representation learning for networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.

[9] Aaron Clauset, Mark EJ Newman, and Cristopher Moore. 2004. Finding community structure in very large networks. *Physical review E* 70, 6 (2004), 066111.

[10] Michael AA Cox and Trevor F Cox. 2008. Multidimensional scaling. In *Handbook of data visualization*. Springer, 315–347.

[11] Edsger W Dijkstra et al. 1959. A note on two problems in connexion with graphs. *Numerische mathematik* 1, 1 (1959), 269–271.

[12] Lester R Ford Jr. 1956. *Network flow theory*. Technical Report. Rand Corp Santa Monica Ca.

[13] Paolo Frasconi, Marco Gori, and Alessandro Sperduti. 1998. A general framework for adaptive processing of data structures. *IEEE transactions on Neural Networks* 9, 5 (1998), 768–786.

[14] Lise Getoor. 2003. Link mining: A new data mining challenge. *SIGKDD Explorations* 5 (07 2003), 84–89. https://doi.org/10.1145/959242.959253

[15] Andrew Goldberg and Renato Werneck. 2005. Computing Point-to-Point Shortest Paths from External Memory. 26–40.

[16] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 855–864.

[17] William L. Hamilton, Rex Ying, and Jure Leskovec. 2018. Representation Learning on Graphs: Methods and Applications. arXiv:1709.05584 [cs.SI]

[18] Peter E Hart, Nils J Nilsson, and Bertram Raphael. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics* 4, 2 (1968), 100–107.

[19] Mohammad Hasan, Vineet Chaoji, Saeed Salem, and Mohammed Zaki. 2006. Link Prediction Using Supervised Learning. (01 2006).

[20] Xiaofei He and Partha Niyogi. 2004. Locality preserving projections. *Advances in neural information processing systems* 16, 16 (2004), 153–160.

[21] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. 1989. Multilayer feedforward networks are universal approximators. *Neural networks* 2, 5 (1989), 359–366.

[22] Ian Jolliffe. 2005. Principal component analysis. *Encyclopedia of statistics in behavioral science* (2005).

[23] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[24] Thomas N. Kipf and Max Welling. 2016. Variational Graph Auto-Encoders. arXiv:1611.07308 [stat.ML]

[25] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436–444.

[26] David Liben-nowell and Jon Kleinberg. 2003. The Link Prediction Problem for Social Networks. *Journal of the American Society for Information Science and Technology* 58 (01 2003). https://doi.org/10.1002/asi.20591

[27] Dijun Luo, Chris HQ Ding, Feiping Nie, and Heng Huang. 2011. Cauchy graph embedding. In *ICML*.

[28] Aleix M Martinez and Avinash C Kak. 2001. Pca versus lda. *IEEE transactions on pattern analysis and machine intelligence* 23, 2 (2001), 228–233.

[29] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[30] Maximillian Nickel and Douwe Kiela. 2017. Poincaré embeddings for learning hierarchical representations. *Advances in neural information processing systems* 30 (2017), 6338–6347.

[31] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. 2016. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 1105–1114.

[32] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 701–710.

[33] Fatemeh Salehi Rizi, Joerg Schloetterer, and Michael Granitzer. 2018. Shortest path distance approximation using deep learning techniques. In *2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, 1007–1014.

[34] Sam T Roweis and Lawrence K Saul. 2000. Nonlinear dimensionality reduction by locally linear embedding. *science* 290, 5500 (2000), 2323–2326.

[35] Ramesh Sarukkai. 2000. Link Prediction and Path Analysis using Markov Chains. *Computer Networks* 33 (06 2000), 377–386. https://doi.org/10.1016/S1389-1286(00)00044-X

[36] Blake Shaw and Tony Jebara. 2009. Structure preserving embedding. In *Proceedings of the 26th Annual International Conference on Machine Learning*. 937–944.

[37] Alessandro Sperduti and Antonina Starita. 1997. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks* 8, 3 (1997), 714–735.

[38] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*. 1067–1077.

[39] Ben Taskar, Ming-Fai Wong, Pieter Abbeel, and Daphne Koller. 2003. Link Prediction in Relational Data. In *Proceedings of the 16th International Conference on Neural Information Processing Systems* (Whistler, British Columbia, Canada) *(NIPS'03)*. MIT Press, Cambridge, MA, USA, 659–666.

[40] Joshua B Tenenbaum, Vin De Silva, and John C Langford. 2000. A global geometric framework for nonlinear dimensionality reduction. *science* 290, 5500 (2000), 2319–2323.

[41] Konstantin Tretyakov, Abel Armas-Cervantes, Luciano García-Bañuelos, Jaak Vilo, and Marlon Dumas. 2011. Fast fully dynamic landmark-based estimation of shortest path distances in very large graphs. In *Proceedings of the 20th ACM international conference on Information and knowledge management*. 1785–1794.

[42] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 1225–1234.

[43] Yiding Yang, Xinchao Wang, Mingli Song, Junsong Yuan, and Dacheng Tao. 2021. Spagan: Shortest path graph attention network. *arXiv preprint arXiv:2101.03464* (2021).

[44] R. Zafarani and H. Liu. 2009. Social Computing Data Repository at ASU. http://socialcomputing.asu.edu

[45] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. 2020. Graph neural networks: A review of methods and applications. *AI Open* 1 (2020), 57–81.