A GAE Approach to Node Embedding applied to a Link Prediction problem

Romy Beaute, Kamil Akesbi, Grégoire Dutot

Abstract—**In this report, we present the main approaches we tested for the MVA Altegrag 2022 Kaggle challenge. The goal is to predict wheter a research paper cites another research paper. We embedded information about the papers thanks to unsupervised framework. We also implemented a GNN encoder to produce more fine-tuned embeddings. These embeddings are finally fed to a Multilayer Perceptron that predicts the probability of two nodes being linked to one another. We achieve quite good performance on the prediction task and observe the efficiency of fixed-size neighborhood sampling to adapt the GAE framework to large graphs.**

I. Introduction

This project aims at applying machine learning techniques to a citation prediction problem. We consider this problem as a classification one : We compute edge information as a concatenation of node features and use it to train a classifier that predicts the probability of two nodes being linked. The overall problem of link prediction can be extremely important in many domains, such as gene-protein interaction or recommendation system. The particularity of our problem lies in the fact that each paper is not only a node in a citation graph, but is also represented by its abstract and its list of authors, that are important features for citation. This leads us to combine NLP and graph machine learning techniques.

Recently, many techniques that seek to make link prediction thanks to decoding local and global structural information have appeared. These techniques usually learn a mapping function to embed nodes as $\mathbb{R}^d$ vectors, which then allows to perform different prediction task, like link prediction. Some of these methods are based on the Skip-Gram method, like node2vec[1] or DeepWalk[2]. Yet a notable drawback of these methods is the inabilty to evaluate them, as they are fully unsupervised. More recent approaches use deep learning methods that can provide each node with strutural representations specific to a particular task. These new methods have achieved state of the art performance in both node classification and link prediction tasks. We will mainly focus on Graph Attention networks (GAT)[3] and GraphSAGE based architectures [4].

While working on this problem, we were confronted to many difficulties emerging its particular structure. A citation shall be represented as a directed link (i.e article citation does not imply citation reciprocity), while most Graph Autoencoders (GAE) techniques assume the graph is undirected. Another issue is the large size of the graph (138,499 nodes and 1,091,955 edges in total), which makes it impossible to load the adjacency matrix in memory. These specificities have constrained our overall approach to the problem and the architecture we developed.

In this project, we propose to implement a whole pipeline that uses the different types of informations contained in each paper and the available citation graph to generate features for each paper. These features are then used to predict the probability of one paper citing another. Our pipeline consists in :

- Retrieving embeddings from each abstract thanks to the Doc2Vec method ;
- Retrieving embeddings from papers' authors by applying the Node2vec method to various coauthor graphs ;
- Applying GNN encoding techniques to generate node embeddings that can help to reconstruct the graph ;
- Using these various embeddings to train a Multi Layer Perceptron on the link prediction task.

Our implementation shows that GAE brings qualitive representation of nodes, which translates in better performance in the prediction task. Moreover, we show that fixing the neighborhood sampling is a good way to adapt Message passing Neural Networks to large graph while preventing overfitting.

The rest of this paper is composed as follows : We first present shortly the techniques we tested in our implementation and the reasons behind their use. We then fully describe the functionning of our pipeline. Finally, we present and discuss the results obtained with our implementation.

II. Methods description

To provide a convenient solution to the problem, the first step consisted in retrieving useful features from the abstract, the authors, and the graph datas to generate different types of embeddings, each of them containing information of different type. To do so, we chose doc2vec to transform each abstract into an embedding vector and node2vec to extract the same type of vectors from the whole citation graph and the author graph. Note that these vectors were not used directly for the link prediction per see, but to initialize the features of our GAE-like architecture.

a) ***Doc2vec & Node2vec embeddings:*** We used the Doc2vec implementation available in the `gensim`[1] library and the `pytorch-geometric` [2] implementation of Node2vec. Both of these models rely on the Skipgram method evoked in class. The algorithm samples trajectories of words (or nodes), then uses these trajectories to estimate the probability of appearance of one node given the embedding of its context. This is done by solving the following optimisation problem :

[1]https://radimrehurek.com/gensim/

[2]https://pytorch-geometric.readthedocs.io/

$$\min_f \frac{-1}{T}\sum_{i=1}^{T}\sum_{\substack{j=i-w\\ j\neq i}}^{i+w} \log P(v_j|f(v_i))$$

where $(v_i)_i$ are the words, f the embedding function and w the window size delimiting the context of each word.

Doc2vec [5] introduces a "document vector", which is added to the context. This is how it trains a document embedding vector. Node2vec [1] uses biased random walks to learn node representation based on their network role and on the community they bleong to.

These two methods are easily implementable, which is why we decided to use them. However, they need quite long training and are purely unsupervised. This makes it difficult to assess their quality. This is why we decided to use some of the embeddings obtained to train a GAE, which gives us a better insight in the quality of the embedding for the link prediction task.

*b) **Graph Autoencoder**:* We saw in class that the graph autoencoder framework provided a good method to produce node embedding. Moreover, as the objective is to reproduce the edges of the input graph, it seems adapted to the link prediction task. Our Graph autoencoder uses a Message Passing Neural Network to encode each node of the graph. For this we chose to use GraphSAGE and Graph Attention Network (GAT) for this encoder.

GraphSAGE : The GraphSAGE [4] model allows us to deal with very large graph, as we can sample a fixed size neighborhood for the message passing. This allowed us to use deeper GNN while maintaining short training time. The message passing of a graph sage model operates as follows :

$$m_v(t) = \text{AGGREGATE}^{(t)}\left(\left\{h_u^{(t)}|u \in \mathcal{N}^k(v)\right\}\right)$$

$$h_v^{(t+1)} = \sigma\left(W^{(t)}[h_v^{(t)||m_v^{(t)}}\right)$$

$$h_v^{(t+1)} = norm(h_v^{(t+1)})$$

Graph Attention Network (GAT [3]) : GNNs can use different type of aggregate operation when exploiting the local graph structure, depending on the type of relational importacne we want to assign between nodes and its neighbors. The main idea of GAT is that the messages from some specific neighbors may be more important than messages from others. In practice, this means that the contribution of each neighboor node in the aggregation have to be weighted according to its importance. To assign different importance weights on each edge (anisotropic operation), GAT applies a self-attention mecanism to get the attention coefficients. For nodes $v_j \in \mathcal{N}(v_i)$, computes attention coefficients that indicate the importance of the features of the node v_i to node v_j. This can be formally written as follow :

$${\alpha_i}_j^t = \frac{exp\Big(LeakyReLu\big(a^T[W^t h_i^t||W^t|h_j^t]\big)\Big)}{\sum_{k\in N_i} exp\Big(LeakyReLU\big(a^T[W^t h_i^t||W^t h_k^t]\big)\Big)}$$

Training the **GAE** is usually done through the edge reconstruction task $\sigma(Z_i^T Z_j)$. This task is easy to compute, but is symmetrical and hence invariant to the direction of the links, which may hinder the representation quality of the encoder. It is also impossible to use this type of reconstruction to correclty predict the existence of a directed edge between two nodes. This is why we train a final MLP using the representation given by the encoder.

To solve the representation quality issue, we also investigated the possibility to use a non-symmetrical MLP predictor to reconstruct the edges. This would transform the reconstruction into $MLPPred(Z_i, Z_j)$. We did not conlude with this approach as it may generate too much overffiting on the graph structure. We also looked into the possibility of directly using this type of predictor to our edge classification task, but difficulties in the pipeline division between training and testing prevented us to implement this idea.

*c) **Neighborhood sampling**:* [Gu et al, 2019] [6] argues that GAT arhitecture is in itself not adapted to link prediction problem since it has to to consider the entire graph, which quickly fills up the memory when considering more than two layers of message passing. Moreover, we empirically observed that the embeddings generated with a GAT based encoder were overfitting on the graph structure, which lead to huge differences between the train and test performances.

As proposed in [6], we chose to consider fixed size sampled neighborhoods to train the GAT layers, as in the GraphSAGE framework. For example, for a 3-layered GNN, we will fix a neighboorhood sampling strategy of $[n_1, n_2, n_3]$, which means that to form or message passing flow graph, we sample n_3 neighbors of the incident nodes, n_2 neighbors of these sampled nodes, then n_1 neighbors of these later sampled nodes. Figure 1 (taken form dgl.ai) shows an example of a $[2, 2]$ neighbor sampling strategy.

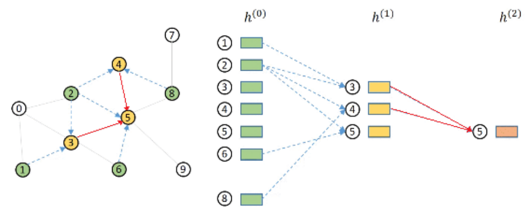

Figure 1: Message passing through neighbor sampling $[2, 2]$

III. Our pipeline

We used the Deep Graph Learning[3] (dgl) library's implementation of GraphSAGE and GAT, as well as its convenient dataloader to develop our full pipeline.

*a) **Abstract embeddings**:* We embed the abstract into vectors of dimension 50 thanks to the Doc2Vec model. Words are tokenized then fed to the Doc2vec architecture, which was initialized with the given conditions :

- varying learning rate α, from 0.025 to 0.00025 ;
- uses PV-DM and hierarchical softmax algorithms
- all words that appeared less than twice are discarded ;

[3]https://www.dgl.ai/

*b) **Author embeddings:*** We construct two different graphs both containing information about the authors of the articles :

- The first one describes co-authors : the authors are the nodes of the graph and edges weights correponds to the number of commun articles between co-authors. It is an undirected graph with 174,961 nodes and 569,033 edges.
- The second one links articles whith their number of commun authors : the articles are the nodes of the graph and edges corresponds to the number of commun authors. It is also an undirected graph with 138,499 nodes and 2,570,106 edges.

We trained Node2Vec to learn node vectors for both graphs and obtained **32** dimension and **64** dimension embeddings.

We used the following hyperparameters:

- Concerning the random walks : the algorithm starts 20 random walks from each nodes, and each walk is of length 30. The window size delimiting the context of each word is set to 10.
- The number of neagtive samples is set to 1.
- We let the parameters **p** and **q** to 1 (default). **p** controls the likelihood of immediately revisiting a node in the walk whereas **q** allows to differentiate between "inward" and "outwards" nodes. These two parameters are essential to control the interpolation between a breadth-first strategy and a deapth-first strategy.

For both embeddings dimensions, we trained the models during 20 epochs on the complete graphs. As we want to learn the best possible embeddings for each node we used all the graph during training (no splitting). We observed a fast convergence (after only 4 to 5 epochs) but kept the training until the end :

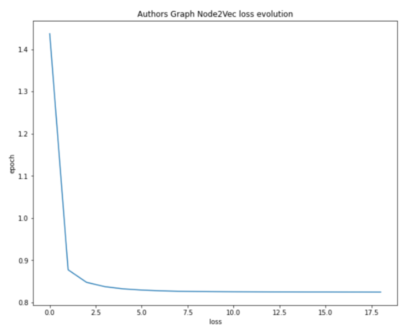

Figure 2: Node2Vec training loss evolution on the co-authors graph - (32d embedding)

Because the training is completly unsupervised, it was hard for us to estimate the quality of the obtained embeddings. One approach was to visually observe how close co-authors vectors where in a low-dimensional space. We used PCA to project the embeddings into a 2D space :

In 4, we can see that the authors *James H. Niblock* and *Jian-Xun Peng* are really close. This is coherent with the fact that they are co-authors (1 article in commun) but also that they have commun direct neighboors : both of them are co-authors with Karen R. McMenemy and *George W. Irwin*. *Karen Rafferty* is located a bit further the 2D space because,

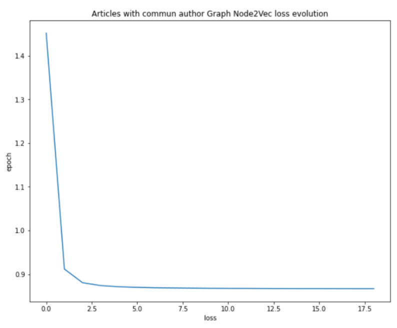

Figure 3: Node2Vec training loss evolution on the articles with number of commun authors graph - (32d embedding)

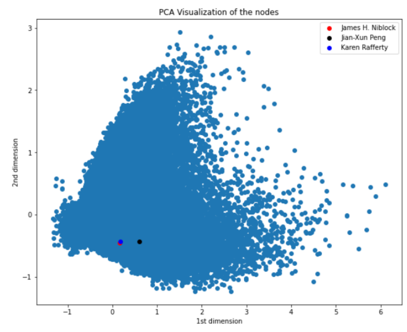

Figure 4: PCA results on the 32-dimension authors embeddings

even if she has been 4 times co-author with *Jian-Xun Peng*, they don't have any other commun neighboor.

To obtain features at the article level using the authors graph, we average the obtained authors embeddings by article.

The embeddings obtained on the second graph are directly associated to the articles (they are the nodes of the graph) and can thus easily be used as features.

*c) **GNN architecture:*** We also used the Node2vec model to produce $\mathbb{R}^{50}$ representation vectors that we used to initialize the features of every nodes before passing in to our GAE architecture. This being done, we then proceed to a 90%-10% split of the edges of the citation graph to form a training and validation set.

As explained in the preious section, we use stochastic message passing updates to embed the nodes of the graph. We compute the message passing graphs flows over the incident nodes of batches of 2048 edges. These graphs flows contains all the edges necessary to compute the sequence of message passing operation. These flows are constructing thanks to an edge sampler that sequentially samples nodes from the neighborhood of a give node. For each batch, we also used uniform negative sampling to get 2048 non linked pair of nodes

The message passing flows are computed on the undirected graph, as experiments demonstrated that for a given model, using undirected graph flows greatly improved the performance of the overall model. We also considered that is was desirable to propagate messages in both directions, hence a cited paper could give insights into the representation of a citing one.

GraphSAGE based model : This model is composed of 3

GraphSAGE layers, each having an hidden vector dimension of 64. ReLU activations are applied at the end of each intermediate layer. These layers use a mean aggregator function. In this model, we considered a $[7, 15, all_{nodes}]$ neighbor sampling strategy.

GAT based model This model is composed of 2 GAT layers, each having a hidden dimension of 64. The first layer has 4 different heads, and the outputs given by each of these heads are concatenated before being passed, after an elu activation, to the second layer. This last layer also has 4 heads whose outputs are averaged to give the final representation. This architecture is directly inspired from [3]. We considered for this model a $[3, 3]$ neighbor sampling strategy. We also tried a fully deterministic neighbor sampling $[all_{nodes}, all_{nodes}]$.

We have tried two types of decoder for this GAE architecture : a simple dot-product one and a more elaborate MLP one (tested on the directed graph). The latter proved to induce too much overfitting in the graph representation, thus leading to bad performance on the test set. This is why we kept the former decoder, which is also simpler and more classical. We then compute the cross-entropy loss using the output of the decoder. We actually use the binary cross entropy loss with logits, as the predictor only computes the dot product.

We trained these different GAE models for 10 epochs each. We use the outputs of each model's decoder as embeddings of dimension 64 for every nodes of the graph.

*d) **Final Classifier**:* Thanks to the previous operations, we have for each paper :

- A Doc2vec based representation of the paper's abstract ($\mathbb{R}^{50}$);
- The Node2vec based representation embedding the coauthoring links of the paper ($\mathbb{R}^{32}$);
- A GAE based representation embedding each paper structural role in the citation network ($\mathbb{R}^{64}$).

For each node, we concatenate all these embeddings, giving vectors of dimension 146. We use another edge dataloader, similar to the one used in the GAE framework, except that in this case we consider edges as directed. We split edges in 90%-10% for validation purpose and also use uniform negative sampling to draw a equal number of non linked node pairs.

Given one pair of nodes (v_s, v_d) (either linked or not), we concatenate the embedding z_s of the source node to the destination one z_s, giving an embedding of the pair $z_{s,d} = [z_s || z_d] \in \mathbb{R}^{2\times 146}$. Note that the order of concatenation will always remain the same, which will allow the classifier function to gain non symmetrical information about the pair of node.

Indeed, we choose this classifier function to be a 3 layerd MLP constructed as such :

- A first layer computing $z_h^{(1)} = ReLU(W_1 z_{s,d})$, with $W_1 \in \mathbb{R}^{64\times 292}$. A dropout with rate 0.5 is applied to this layer.
- An intermediate layer computing $z_h^{(2)} = ReLU(W_2 z_h^{(1)})$, with $W_2 \in \mathbb{R}^{64\times 64}$. This layer also has a dropout rate of 0,5.
- An final layer computing $l_p = W_3 z_h^{(2)}$, with $W_3 \in \mathbb{R}^{1\times 64}$ which gives the log-probability of the v_s being connected to v_d.

This classifier is trained for 10 epochs using a Binary cross-entropy loss with logits. We used an Adam Optimizer with a learning rate of 0.0005 It is used as such to make the final predictions on the test set provided.

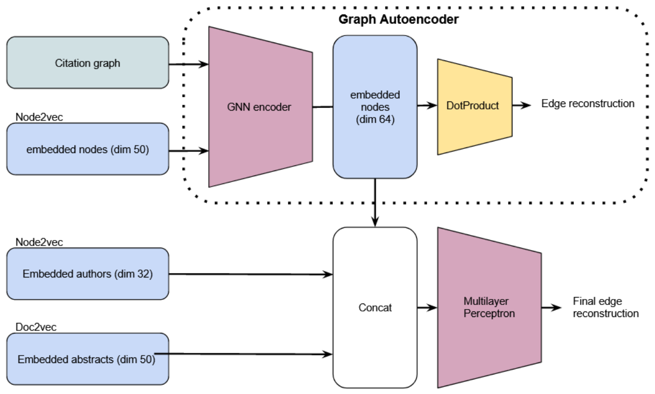

Figure 5: Architecture of the entire pipeline

IV. Results and Discussion

The results of our two approaches (GraphSAGE and GAT based) are shown in Table I. On this table, the computation time is divided between the GAE training and MLP training. Figures 6 and 7 respectively show the training process of the GAE and final MLP classifier for both approaches (only the [3, 3]-GAT is shown). The score represented is always the binary cross-entropy score.

Approach	Train BCE	Val BCE	Test BCE	Time
Graph baseline			0.480	1 min
GraphSAGE based	**0.135**	**0.136**	0.212	18 + 4 min
GAT based (deterministic)	0.093	0.094	0.237	40 + 4 min
GAT based [3, 3]	0.188	0.186	**0.198**	8 + 4 min

Table I: Scores and computation time for each approach

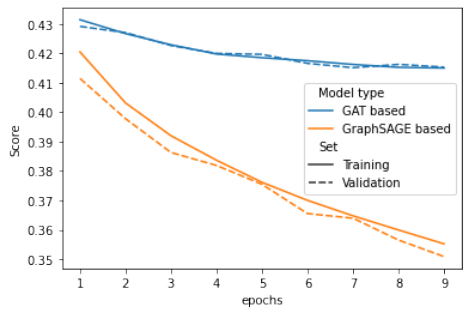

Figure 6: Cross entropy score during the GAE training process

We note that the neighborh sampling strategy greatly accelerates the training process. Yet the GraphSage architecture allows to go deeper without too much additionnal computation time, as it involves fewer computations.

It appears from Table I that a GAT based GAE with deterministicyields more precise embeddings for the graph reconstruction task. This confirms the efficiency of the GAT

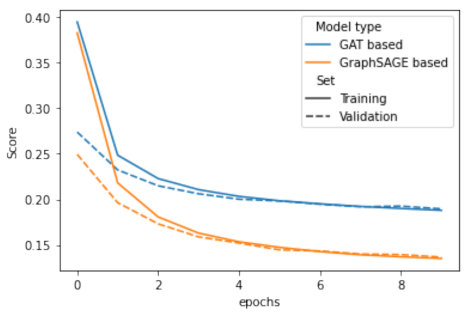

Figure 7: Cross entropy score during the final classifier training process

architecture that has already been observed in many papers [3]. These embeddings help in constructing a better classifier during the final phase.

Overall, we notice that all models show great differences bewteen their training and test score. Our most plausible explanation for the moment is the fact embeddings formed by the GAE training are overfitting on the structure of the graph. This might be observed in Figure 6. Indeed, all available edges are used for the message passing in the GAE training step, which might induce data leakage. This leakage makes it more difficult to evaluate the architecture true performance.

It has proven to be difficult to solve this issue without fully modifying our whole pipeline. An efficient solution is to take a more random sampling strategy, as used with the GAT-[3, 3] model. This model achieves less precise edge reconstruction performance for both training phases, but proves to have a closer and better test score. This overfitting of the GAE is the most noticeable inconvenience of our method and we lacked time to investigate it further.

The random nature of GraphSAGE message passing seems to allow the method to partially avoid this data leakage. Overall performance of all methods are still quite good in comparison with the baseline.

The graph of Figure 6 and 7 indicates that models could be trained for more epochs, as validation and training scores are still similar after 10 epochs. This means that performance may still be increased with a proper tuning of the models.

We looked into various modifications to tune our models, but it appeared that the most impactful ones consisted in changing the embeddings. For example, we tried to use directly the embeddings without the GAE framework (as another mean to avoid the partial data leakage), but the performances obtained were grealty diminished (even though they were still better than the baseline). This proves that our GAE architecture creates qualitative structural representations.

V. Conclusion

In this project, we applied a GAE based approach to produce embeddings that help solving the link prediction problem. The particularity of the graph studied (large and directed) led us to use a stochastic training of the GAE and to develop a final classifier that could extract non-symmetrical information from node pairs.

Qualitative embedding appeared to be the main driver of link prediction performance and the GAE approach proved to provide better embeddings than the basic node2vec model.

A noticeable problem of our method is its lack of transparency. Every embeddings are produced with black box models, such as GAE, Doc2vec and Node2vec. This made it difficult to evaluate the drivers of performance of our architecture. Moreover, we can imagine that a better analysis of textual and author data could provide much more qualitative information and ameliorate our architecture.

Overall, we familiarized with the GAE framework for large graphs, and got our hands on GNN mechanisms and NLP embeddings techniques. The DGL library has proven to be really useful in our work.

References

[1] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," *CoRR*, vol. abs/1607.00653, 2016.
[2] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk," *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, Aug 2014.
[3] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," 2018.
[4] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *CoRR*, vol. abs/1706.02216, 2017.
[5] Q. V. Le and T. Mikolov, "Distributed representations of sentences and documents," *CoRR*, vol. abs/1405.4053, 2014.
[6] W. Gu, F. Gao, X. Lou, and J. Zhang, "Link prediction via graph attention network," 2019.