


---

# RELATIONAL GRAPH NEURAL NETWORKS

---

NANTES UNIVERSITÉ / POLYTECH NANTES

**POURTAUD Robin**  & **DOLORES Maxime**

Supervised by Le Capitaine Hoël 

2022

## Abstract

For the last 15 years, Graph Neural Network has been used to perform Deep Learning tasks on Graph Structured data. The original GNN was used on a single relation and undirected graph but their properties show some potential to be updated. Graph Convolutional Network GCN and Graph Attention Networks GAT improve considerably their performance, and a lot of new framework added the possibility to consider relations, edges features or node features. To evaluate GAT and GCN, we perform these models on an open education dataset x5gon. We eventually propose a new model : S-GAT. Based on relational-GAT we took a step further by using the semantic aspect of vertices and edges features simultaneously. Recent applications of the Graph Attention Network (GAT) presenting good results by enhancing certain relations being more important, it seems to be suited to take the semantic aspect into account. To capture the semantics concept, S-GAT uses Doc2Vec and K-means to cluster similar nodes. Further research and experiment are needed as proof of concept to evaluate S-GAT.

## Keywords

Graph Attention Network, Graph Convolution, Link prediction, Semantic Embedding

## CCS Concept

Computing methodologies → Neural networks

## 1 Introduction

Graphs are a great way to model all kinds of data such as transport networks, user ratings or even the way a disease spreads, just to name a few. In this project we will focus on working with open educational resources from X5GON's dataset with the goal of associating a resource with Wikipedia concepts.

X5GON is a platform created from a European project that aims to provide a better learning experience by listing open-sourced class materials in different formats (PDF files, videos lectures, slides, ...) and different languages from across Europe.

The database can be represented as a bipartite graph with two types of nodes :

- **Resources** : also called materials throughout the article, they correspond to content that can be used by teachers in their classes or by students to learn from.
- **Concepts** : they correspond to Wikipedia concepts that appear in the different resources of the dataset.

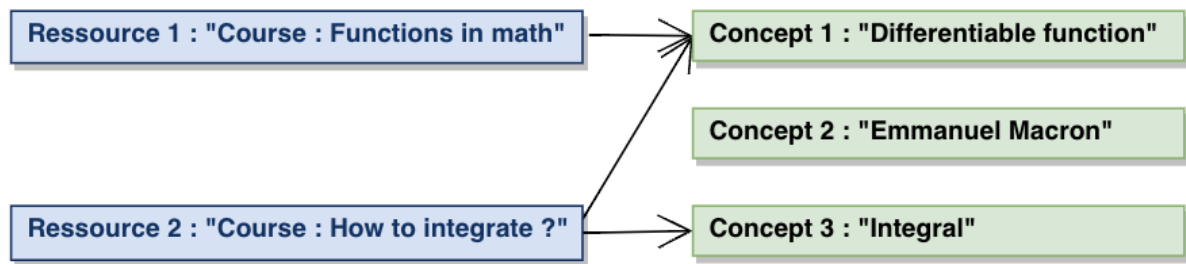


Figure 1: Example of resource-concept relation

Figure 1 illustrates how the data would look like in a graph format. The different edges<sup>1</sup> signify that there exists a relationship between a resource and a concept. In a more concrete way that means that the concept is mentioned in the resource.

Our goal in this paper is to predict if a relationship should exist between a given concept and a given resource.

We will eventually propose a new theoretical approach to solving these kinds of problems that we have not been able to test or implement within the limited time we were given.

## 2 State of the art

In 2009, Graph Neural Networks (GNN) [Sca+09] introduced a new type of neural network model that can handle graphs as an input. This approach worked with cyclic graphs, digraphs, undirected graphs and showed promising results at the time.

Graph Neural Networks are perfectly suited to work with large datasets while preserving their structure. But what is probably the most important thing they bring is that thanks to their topological properties, they make it possible to build neural network models with heterogeneous data.

The whole idea behind GNN is to build a model using an encoder and decoder structure to work in a lower dimension space called the embedding space. During the encoding process, for each node of the graph, the model tries to build the best possible representation of the node in the embedding space. Then in the decoding process the goal is to use the node's representation in the embedding space to determine the model's output.

To compute a node's representation, the GNN model aggregates information from the node and its neighborhood. Later, this approach was adjusted to solve graph classification tasks [Yin+18] [Xu+19], node classification tasks and link prediction tasks [Zha+20].

Graph Convolutional Networks (GCN) [KW17] were introduced in 2017 and greatly outperformed the original approach. The Graph Convolutional Network approach proposes the following changes :

- Unlike in the GNN approach, here the representation of a node does not include itself, which means that the information is computed only by aggregating features from the node's neighborhood. Note that nodes with self-loops will include their own features when computing their representation.
- GCN uses the symmetric normalization of the Laplacian. This could potentially lead to the vanishing gradient problem. It occurs during the training phase, the model might make small weights vanish which can lead to the model not learning anything new. [Pro]
- To solve the vanishing gradient problem a "re-normalization trick" is performed.

Note that GCN does not support edge features or relation types natively which makes it hard to perform any link classification tasks.

For example, a very common problem when it comes to graphs is the completion of knowledge graphs (KG). Knowledge graphs are a type of graph used to represent relations between resources on the internet. They are based on triples which are 3-tuples composed of a subject, a predicate and an object.

With this implementation of GCN it would be impossible to work with the predicate element of triples.

A year later, GCN was generalized to work with digraphs containing multiple relation types [Sch+18] which enabled tasks such as knowledge graphs completion. A lot of work has been conducted on solving this problem, recently TransE [Cha+21] and RotatE [Sun+19] have proposed new approaches with very convincing results. However these methods are not based on GCN and suffer from the information not propagating in the graph.

VR-GCN[Ye+19], TransGCN[Cai+19] and CompGCN[Vas+20], are using a graph neural network

<sup>1</sup>Edge / Link / Relation / Connection and Node / Entity are used in an interchangeable way throughout this article

to learn multi-layer latent knowledge from the embedding space for both entities and relations. In these networks, the relation does not propagate via convolutions operations, leading to sub-optimal solutions.

A model that tries to solve the different problems that we just explained is Knowledge Embedding Based Graph Convolutional Network (KE-GCN) [Yu+20].

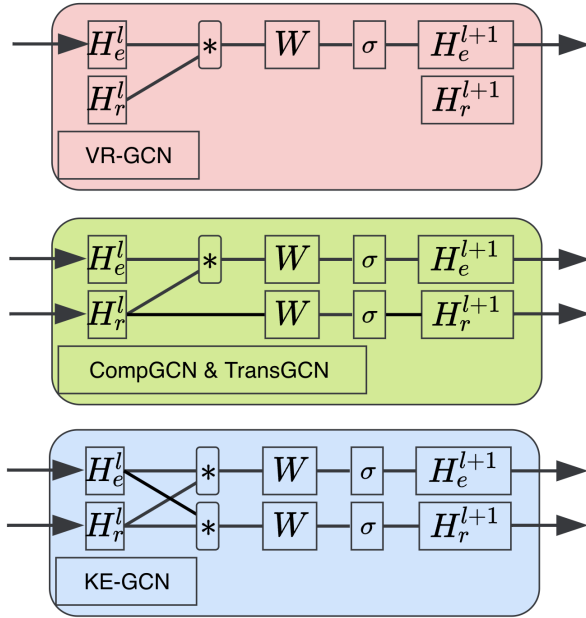


Figure 2: Visualizing the different models [Yu+20]

- \* represents an aggregation function (concatenation, average, ...).
- $\sigma$  represents the activation function (sigmoid, LeakyRelu, ReLU, ...).

KE-GCN updates both the entity and relation embeddings using graph convolution operations leveraging various knowledge embedding techniques. This approach gives better results in entity or relation alignment tasks and knowledge graph entity classification tasks. KE-GCN outperforms VR-GCN, and KBGAT [Xie+20]. Among all graph embedding methods used TransE [Cha+21], TransH[Wan+14], Dismult[Cha+14], TransD[Ji+15], RotatE[Sun+19], QuatE[Zha+19]. QuatE is the one that has had the best results overall. It shows the importance of using entity representations while updating relation embeddings.

Other alternatives exist like GraphSAGE[HYL17]. Unlike GCN, GraphSAGE aggregates neighbors randomly. However, the randomness is not suited to predict relation types because the different relation types are often not distributed uniformly.

Another approach to work with edge features is the Graph Attention Network (GAT)[Vel+18]. It has sim-

ilar or better results than vanilla GCN or GraphSAGE. GAT was presented in 2018 as a novel convolution-style neural network operating on graph-structured data, leveraging masked self-attentional layers. Without requiring any kind of costly matrix operations (such as inversion). It makes it possible to specify different weights for each node in a neighborhood. As GCN, GAT uses self-connections in order to perform a "self-attention". A graph attention layer takes a set of nodes (with their features) as an input and outputs a new set of nodes. The Attention mechanism used is a single-layer feed-forward neural network. This Attention allows each node to consider each neighbor node differently, unlike GCN.

GAT solves two problems presented in the GCN paper.

- **Memory Requirement :** GCN requires costly matrix operations. GAT does not require matrix decomposition, thus no matrix inversion. However, applying the multi-head attention mechanism multiplies the storage and parameter requirements by a factor of  $K$ , while the individual heads' computations are fully independent and can be parallelized.
- **Limiting assumptions :** Self-connections and edges to neighboring nodes are equally important in GCN. With GAT, adding a different attention to all neighbors edges answer this limit of GCN.

The original GAT convolution is the following :

$$h'_i = \sigma \left( \sum_{j \in \mathcal{N}(i)} \alpha_{i,j} \times W \times h_j \right)$$

With :

- $\sigma$  : the LeakyRelu function
- $\alpha$  : the attention Mechanism
- $h_j$  : the node j in the embedding
- $h'_i$  : the new value of the node i in the embedding
- $\mathcal{N}(i)$  : the neighborhood of  $i \in \mathcal{E}$

In the original GAT paper, the attention mechanism [Vas+17] is represented by the following :

$$\alpha_{i,j} = \text{softmax}(a(W_e \times h_i, W_e \times h_j))$$

With "a" representing the attention function. It is originally designed using a single feed-forward neural network. The following figure represents this neural network. For each relation, the attention is computed

from the concatenation of the embedding of  $i$  and  $j$  :

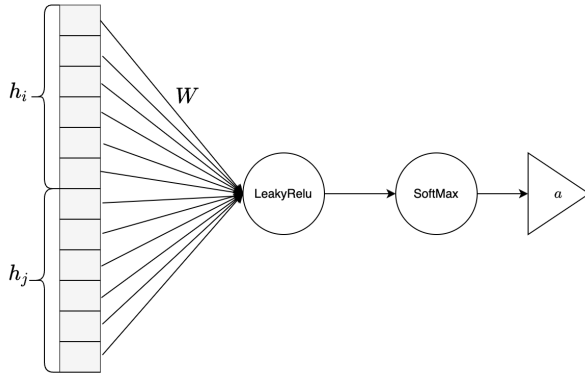


Figure 3: Single Feed-Forward Neural Network

In the publication, the researcher specifies that the computation of  $a$  could be defined with other models, single feed-forward network only acts as a working example.

Moreover, GAT researchers propose a multi-attention concept to stabilize the learning process, similarly to *Attention is all you need* [Vas+17]. To be more precise, the learning process is executed  $K$ -times independently and then the computed features are concatenated :

$$h'_i = ||_{k=1}^K \sigma \left( \sum_{j \in \mathcal{N}(i)} \alpha_{i,j}^k \times W^k \times h_j^k \right)$$

On the last layer, the concatenation could be normalized with  $K$  to present coherent output results :

$$h'_i = \sigma \left( \frac{1}{K} \sum_{k=1}^K \sum_{j \in \mathcal{N}(i)} \alpha_{i,j}^k \times W^k \times h_j^k \right)$$

Figure 4 summarizes the graph attention network with multi-head attention. In the first layer, the induction is initialized using  $h_i = e_i$ . The blue arrows are relations from the first channel ( $k = 1$ ), the pink ones are from the second channel ( $k = 2$ ).  $K = 2$ . From the original graph, a self attention has been added leading to the existence of  $\alpha_{1,1}$ . All of these channels accumulated in  $h_1$  will be summed if  $h'_1$  is the last layer, concatenated otherwise. The induction is performed using the following formula,  $h_i := h'_i \mid \forall i \in \{1 \dots |E|\}$

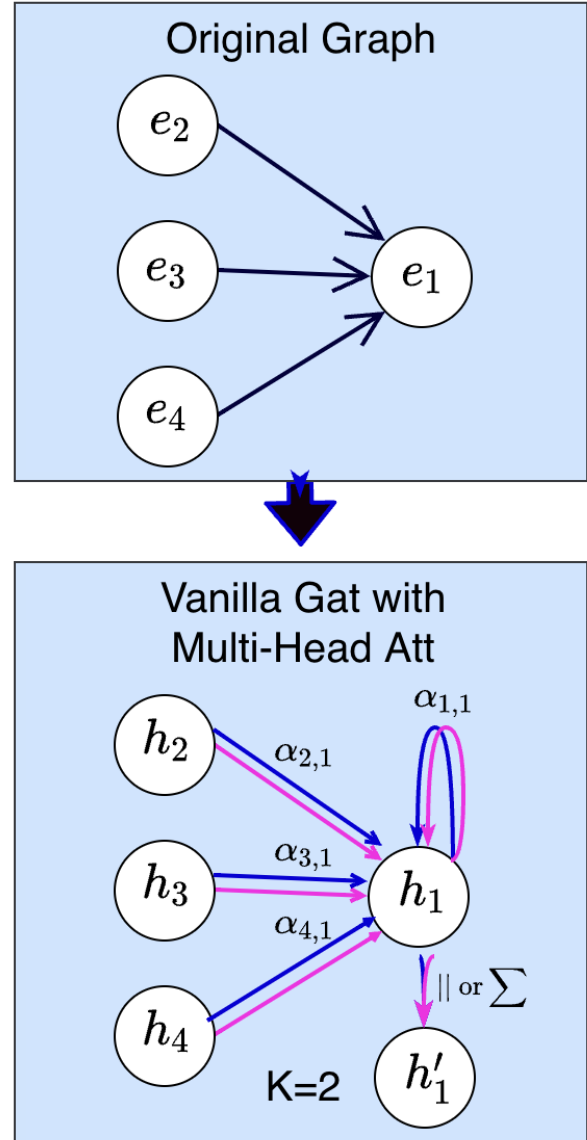


Figure 4: Graph attention network with head=2

Currently, even though edge features usually give great insights on the data most papers are not focused on working and experimenting on those.

Edge Graph Neural Network (EGNN) [GC19] is one of the few approaches that tackles this problem. EGNN is declined in multiple models : EGNN(A) and EGNN(C) (A stands for attention while C for Convolution). Each declination presents its own strength and weaknesses.

EGNN(A) proposes a new attention mechanism to include multidimensional non negative-valued edge features on top of the Graph Attention Network. In this new attention mechanism, feature vectors are aggregated from the feature vectors of the neighboring nodes. The attention coefficients are used as new edge features for the next layer :

$$e_{i,k} = \alpha_{i,k}$$

In this context,  $e_{i,k}$  is the sub-vector of  $e_i$

The induction formula defining the aggregation operation is defined as follows :

$$h'_i = \sigma(\|_{k=0}^n \alpha_{i,k} \times W_i \times h_i)$$

Thereby :

$$h'_i = \sigma(\|_{k=0}^n e_{i,k} \times W_i \times h_i)$$

By doing so, EGNN adapts the edge features across the network layers, which helps to capture essential edge features as determined by our new attention mechanism. EGNN(C) layer is derived from EGNN(A) layer. In EGNN(C), the attention is simply removed and replaced by the original edge features vector as such :

$$h'_i = \sigma(\|_{k=0}^n e_i \times W^l \times h_i)$$

These two networks have been used on Cora, CiteSeer and Pubmed networks and have led to better results than GCN or GAT. On the Cora dataset, GAT has 72.01.2% of accuracy against 88.80.3 for EGNN(C). These results were computed with a dense splitting (60%, 20% validation 20% test). However, in some cases, ignoring multidimensional edge features can lead to sub-optimal results. The quality of edge features can increase the performance, and sometimes alienate the model. The analysis and comprehension of the semantic related to edge features could be important to improve this model.

Relational Graph Attention Network (r-GAT) [xc] also tried to solve this problem. r-GAT uses the same multi-head attention mechanism as GAT but the motivation is not the same. In GAT, this mechanism is, as we said earlier, used to stabilize the learning process. In contrast, r-GAT does not use averaging on the output but uses this new mechanism to preserve the entity features in multiple isolated channels having the same semantic topic.

This time, The attention mechanism is calculated from the concatenation of  $h_i, h_i, h_j$ .

$h_r$  is the relation features vector :

$$\alpha^k = \text{softmax}(a(W_e^k \times h_i, W_r^k \times h_{r,i}, W_e^k \times h_j))$$

Unlike EGNN(a), r-GAT does not update  $h_r$  and is always equal to  $r_i$ , the relation vector :

$$h'_i = \|\|_{k=1}^K \sigma(\sum_{j \in \mathcal{N}_i} \sum_{r \in R_{ij}} \alpha_{i,j}^k h_j^k \times r^k)$$

Thanks to the self loop,  $h_i$  is not needed in the formula.

The semantic aspect for r-GAT has been captured thanks to the Query-Aware Attention Mechanism (Qatt). Given a query  $q$  and a subject entity  $s$ , before inputting the entity embedding to the decoder, the attention has been applied between the query and the K components.

Furthermore, the query-aware attention mechanism can also be extended to a multi-head form to stabilize the learning process.

The researchers wanted to show that query-ignorant is a problem to identify more important neighbor during the neighborhood aggregation process.

EGNN(a), EGNN(c) and r-GAT presented state of the art results, however, we could not compare our model to theirs as their code was unavailable at the time of our experiments.

r-GAT and EGNN are, at the moment a relatively light-weight algorithm. A little more intensive than both GAT and GCN but they produce great results nonetheless.

### 3 Experiments

In this section we detail our approach to solving a link prediction problem between open educational resources and Wikipedia concepts.

The dataset we used for our experiments has been constructed using X5GON's network of open educational resources spread among different languages and mediums.

#### 3.1 Dataset

The first step is to build a proper dataset, to do that we pull data directly from X5GON's API, by doing that we can retrieve a list of materials and for each material a list of concepts associated with it. We decide to limit ourselves to english materials only.

Each material comes with some additional data but we are only interested in the material's title and text content.

Then for each material we go through the list of Wikipedia concepts associated with it and for each one we retrieve the page's title and the page's text content using Wikipedia's API.

We obviously also keep track of the links between each material and concept. In the end our dataset is composed of 1000 materials and 7683 concepts, with 16242 links between materials and concepts. Keep in mind that one concept can be linked to multiple materials.

#### 3.2 Data transformation

To process the data in our model we need to transform our text data into numerical data, in order to achieve this we can make use of the Doc2Vec algorithm.

We run Doc2Vec on the concepts and materials separately to reduce both datasets into vectors of 128 numerical features.

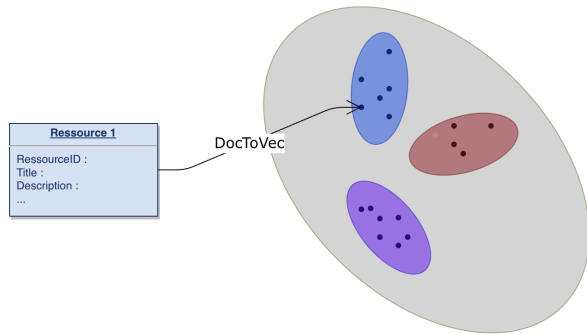


Figure 5: Resource embedding with Doc2Vec

### 3.3 Heterogeneous dataset

Until now both the materials and concepts are separated in different files. In order to use them in our model we need to merge them into a graph. To do that we are using PyTorch Geometric's heterogeneous data handler which enables us to easily build our graph.

Once the graph is built we first transform it into an undirected graph by using a function provided by the library. Then to train our model we take our newly obtained graph dataset and split the links into three datasets, a training one representing 80% of the links and a validation one and a testing one that both represent 10% each. While splitting we also use a parameter of the RandomLinkSplit function which allows us to perform negative sampling with a ratio of 1.

Negative sampling takes the number of links labelled "existing links" and creates an equal number of "non-existing links". Doing this allows us to train our model as if it were a classification task where the goal is to separate correct links that should exist from incorrect links that should not be here.

### 3.4 Model

Our model is a classification model estimating the probability of a link existing between a given concept and a given material. It is composed of an encoder and a decoder.

Firstly the encoder plays the role of reducing the input data to a lower dimension, we call this new representation of the data the embedding space. It represents latent knowledge of the input, as an example in the embedded space we could compute distances to find some similarities between two inputs. In our model the encoder is composed of 2 SAGEConv layers.

Then once the data has been reduced to the embedding space the decoder comes into play, it takes a vector from the materials embedding space and a vector

from the concepts embedding space concatenates them together and extracts the latent knowledge using what it has learned in training to give out its prediction in terms of probability using a Softmax function on 2 neurons as the output.

### 3.5 Results

We compared our model with other graph convolutional models such as GCN, GATConv, ... and here is a table summing up the results :

Model	Layers count	Min loss
SAGEConv	1	0.61897
SAGEConv	2	0.64379
GATConv	1	0.65642
GATConv	2	0.68678
GraphConv	1	0.53214
GraphConv	2	0.77310

The "layers count" column represents the number of layers of the model in the encoder, the "min loss" one corresponds to the minimum cross-entropy loss obtained on the testing set over 300 epochs of training.

If we had to keep one approach over the others it would be SAGEConv as it is the encoder approach that gives us the best results while still having a loss curve that looks decent. GraphConv's loss curve has too much noise and looks abnormal which is why we chose SAGEConv over GraphConv.

We can also make a table of our 1-layer SAGEConv model's results to see the effect of the embedding space's dimension on the loss.

Model	dim(ES)	Min loss
1 layer SAGEConv	8	0.64241
1 layer SAGEConv	16	0.60268
1 layer SAGEConv	32	0.61897
1 layer SAGEConv	48	0.60667
1 layer SAGEConv	64	0.59911
1 layer SAGEConv	72	0.61467
1 layer SAGEConv	96	0.61159
1 layer SAGEConv	128	0.71215

This table shows us that the sweet spot for our case is likely around an embedding space of dimension  $n=64$  which represents a halving of our initial input vectors that had a dimension of  $n=128$  features in the beginning.

If we had to retain one model from all of that it would be the one using a 1-layer SAGEConv encoder with an embedding space of dimension 64. The following figure illustrates the evolution of the cross-entropy loss per epoch for this model.



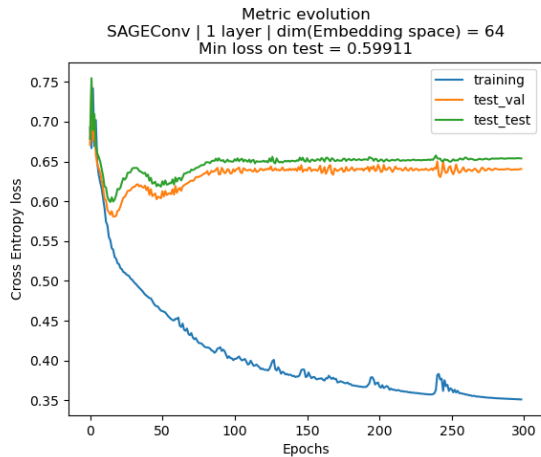


Figure 6: Evolution of the loss over 300 epochs

As we can see on our testing set the loss starts at 0.75 and then slowly decreases to reach approximately 0.60 and then rises again to 0.65 as our model probably starts to overfit and stagnates for the rest of the training. The

optimal number of epochs to train our model on would be around 10 to 15.

### 3.6 Other uses

We can extract data directly from the embedding space to use as a latent knowledge representation in other models or as a way to visualize and plot the different resources and concepts in a lower dimension.

## 4 Proposition and future researches

Our current model predicts the likelihood of a link existing between a resource and a concept. This could be used in a recommendation system to suggest new resources after an user has finished reading a material. Still, it is not ideal. During the training, we ignore an important aspect, the graph is bipartite. Thus, several strategies are available for us to improve the usefulness of the model.

We could predict oriented relation between resources directly as such :

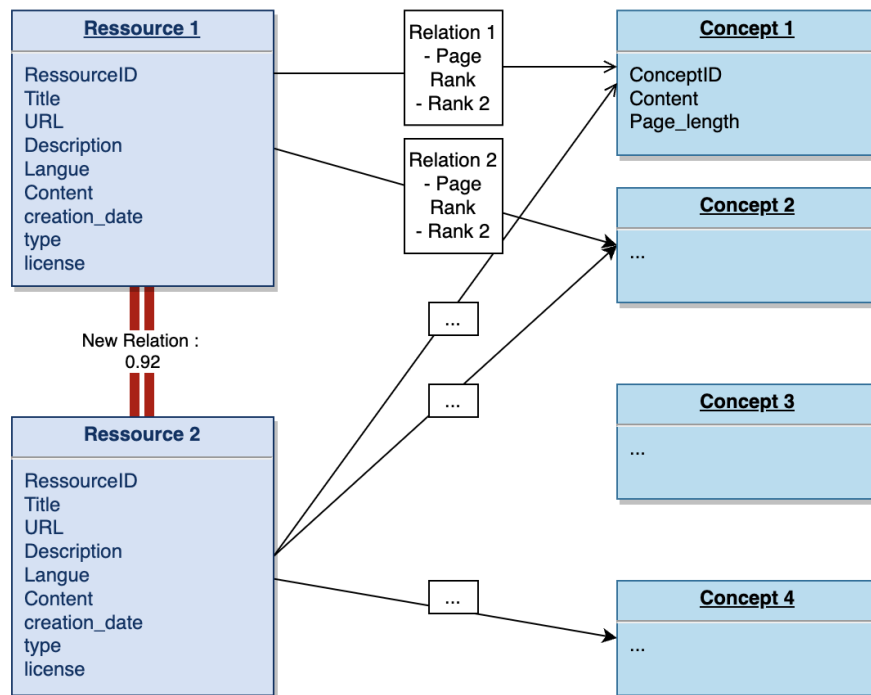


Figure 7: Single Feed-Forward Neural Network

Our current dataset only contains relations between resources and concepts making it impossible to predict the existence of a link between two resources or two concepts. However predicting such links could be a way to improve our model if we wanted to perform more complex link prediction tasks. We could use an algorithm that links the resources with the most common concepts. This

simple algorithm however neglects some important aspect like the resources, relations and concepts features (only identifiers and triples are considered). To obtain a better outcome, we could consider the following classification task, using GAT, we could compute resource embeddings by aggregating information directly on concepts and relations. Then we could compute the distance between

different resources in the embeddings and build new relations based on that.

#### 4.1 Introducing S-GAT

Illustrated by the previous perspectives and future possibilities, we propose a new graph neural network model Semantic-GAT (S-GAT) in this section.

S-GAT is a model optimized to work on bipartite graphs with a latent semantic aspect.

To illustrate the S-GAT, we consider, in this section, a network containing two types of nodes  $a \in A$  and  $b \in B$  such as  $G = (V, E)$  with  $V = AB$ . Edges are a subset of  $A \times B$ . Each Node and each edge possesses features as such :  $\phi(a_1) = \vec{a}_1$ . In practice, each node set of features is contained in a tensor  $H$  and edge feature in an adjacency tensor  $R$  (an adjacency matrix with a vector for each value).

Our goal is to determine the set of relation between two nodes  $(a, a') \in A \mid a \neq a'$ .

##### 4.1.1 Prerequisite

First of all, the format of the data is important. Each set of edge and node features needs to be a vector of the same type. The length of every vector needs to be the same :  $|a| = |a'| \mid \forall a, a' \in A$  (in the same manner for  $B$  and  $E$ )

A distance is used to compute the similarity between 2 nodes. Two nearby vectors should be semantically similar. If the node vectors are not suited to this manner, the model will lead to incoherent results. The simplest way to embed words or text should be using doc2vec or word2vec.

##### 4.1.2 Capture initial semantic channels

At this state, using K-means or any distance unsupervised clustering algorithm between nodes from  $A$  is possible. Node features are used and could in some cases be sufficient. However, many properties, like the edge features or the graph's structure are omitted. We can think of our problem as an example, two courses (resources) could be taught in a contiguous way (one after another) but the two resources have very distinct text. K-means would not be able to cluster together these two courses. If these two courses are linked to a same concept (like the author name), then a graph neural network algorithm would be able to handle the problem. S-GAT will try to tackle this problem.

Inspired by r-GAT, we need to first capture initial semantics channels : K-means is suited to this task. In a very diversified network, K needs to be consequent, though, a high value for K will probably make this model useless. On the other hand, in a very similar network, K needs to be very small. K will have a huge impact on the performance of the network. If K is too small, then the whole dataset could neutralize itself with too much iteration, op-

posites concepts will be in the same channel and could scramble features. If K is too big, then similar concept could be in isolated channels and couldn't interact with each other.

##### 4.1.3 Aggregation operation

First we define the attention function in a similar fashion as r-GAT :

$$\alpha_{a,i,b}^k = \text{softmax}(a^k(a_i || r_i^k || b_i))$$

Such as GAT,  $a^k$  is a single feed-forward network for k-th channel.

This model has not been tested, thus, we propose two convolution mechanisms to study in future research.

##### Updates "a" nodes and "b" nodes

We can update for all nodes, a and b there embedding at each epoch.

$$h'_i = \sigma(|_{k=1}^K \sum_{j \in \mathcal{N}(i)} \alpha_{i,j}^k \times W^k \times h_j^k)$$

However, for non-optimal "K" (a too-small K), as said earlier, it could neutralize the network. Each node, except isolated ones, will tend to be the same.

##### Updates "a" nodes only

To avoid this problem, we can aggregate features only on  $a \in A$  nodes.

$$h'_a = \sigma(|_{k=1}^K \sum_{j \in \mathcal{N}(i)} \alpha_{i,j}^k \times W^k \times h_j^k)$$

Nodes with shared concepts will then be much closer to each other.

It is possible that limiting the propagation of the aggregation lead to a sub-optimal solution.

The last layer should replace the aggregation function by a summation one.

##### 4.1.4 Clustering Task

We suppose that at this point, all A nodes gains a lot of information from the network, respecting the graph structure.

We want now to establish relations between nodes that belong to a.

##### Simple cluster

K-means could be used as an unsupervised algorithm to determine clusters of similar nodes. If we want to propose 10 suggestions, then we need to have  $K = \frac{|V|}{10}$ .



There are multiple issues with this approach. Very isolated nodes could be clustered with a totally different node leading to absurd results. The number of suggestion is very rigid. 2 nodes from 2 close clusters could not be predict to be similar.

Another possibility is to use cosine similarity or any distance computation between each node to pick the  $n$  closest nodes. The complexity of this algorithm is not negligible :  $O(|A|^2 \times n)$


#### **Define an order relation on nodes**

There are few possibilities to define a total order on nodes if there are none originally. In specific datasets, we can make the assumption that less connected nodes are less popular or more difficult to apprehend, thereby, each new relation predicted can be oriented from the less connected to the more popular one.

## **5 Conclusion**

Recent research has shown the possibilities offered by neural graph networks with excellent results. However, the bipartite aspect of semantic graphs is often ignored. In this paper, we tried existing algorithms on our dataset x5gon, a bipartite graph, to evaluate their pertinence and efficiency. Then, we proposed a new semantic graph attention network (s-GAT) for multiple relations, edge features, and node features for bipartite graphs. Further research and experiment are needed as proof of concept to evaluate S-GAT.

## **6 Acknowledgment**

We would like to thanks Hoël Le Capitaine  who helped and supported this research project.

## 7 Appendix

### 7.1 Notation

- $G = (\mathcal{V}, \mathcal{E})$  : A graph is a pair of the vertices and edges set, with  $\mathcal{E} \in \mathcal{V}^2$
- $\mathcal{N}(i)$  : The set of "i" neighbors vertices.
- $A$  : The adjacency Matrix
- $I_n$  : The identity matrix of dimension  $n \times n$
- $|\vec{a}|$  : The length of the vector  $A$
- $\phi$  : An isomorphism associating a node to its feature
- $\parallel$  : The concatenation operation for vectors.
- $\parallel_{k=0}^n p(k)$  : The iterate concatenation of all  $p(k)$   
 $\forall k \in \{0 \dots n\}_{\leq N}$
- $R$  : Is the set of relations.
- $W$  : Is a weight matrix
- $h'_i$  : Is the new embedding of  $h_i$ . It can also be written  $h_i^{(l)} = \dots h_i^{(l-1)} \dots$
- $\star$  : One-to-one multiplication

### 7.2 Metric evolution

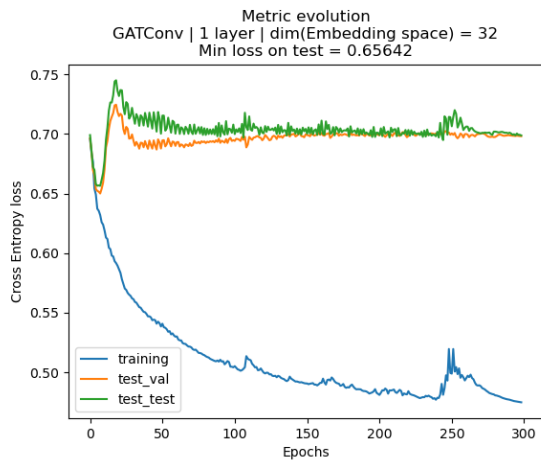


Figure 8: GAT Conv 1 layer dim 32

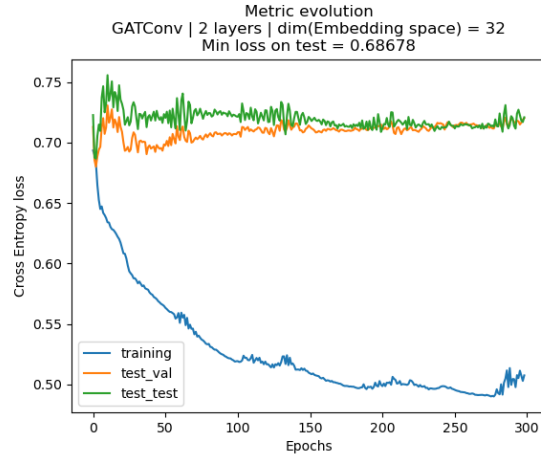


Figure 9: GAT Conv 2 layer dim 32

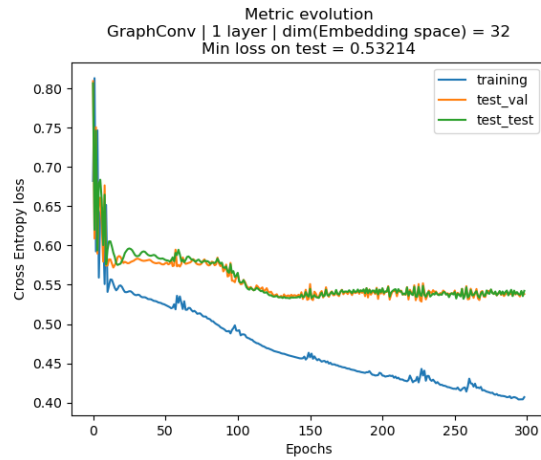


Figure 10: GNN Conv 1 layer dim 32

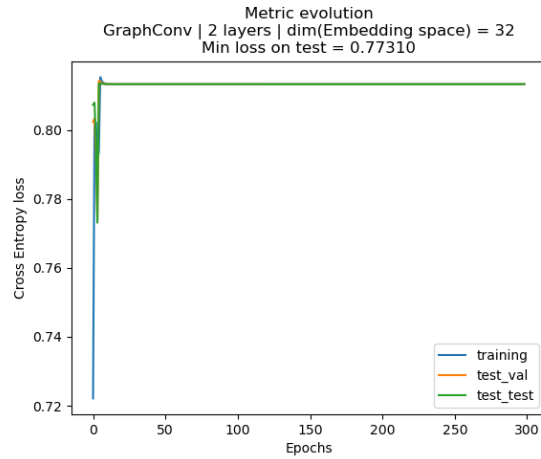


Figure 11: GNN Conv 2 layer dim 32

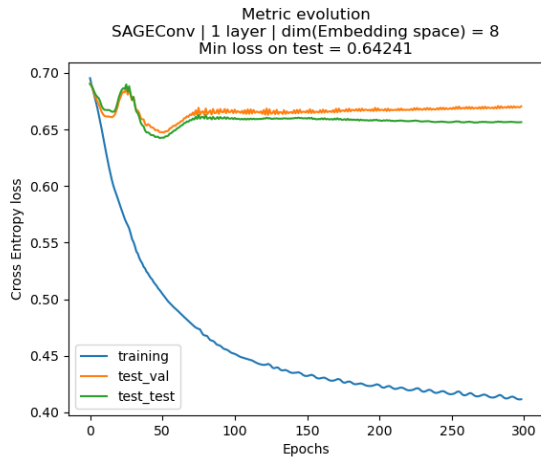


Figure 12: SAGE Conv 1 layer dim 8

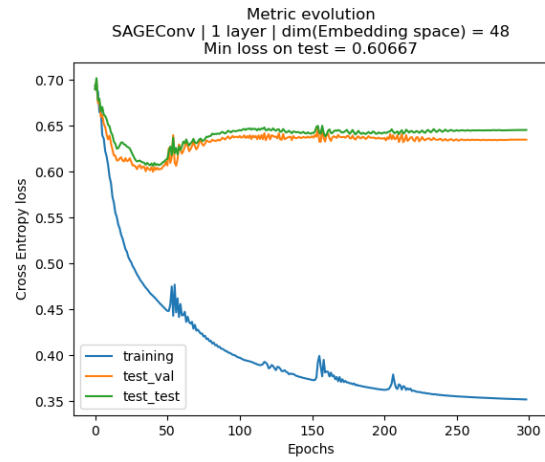


Figure 15: SAGE Conv 1 layer dim 48

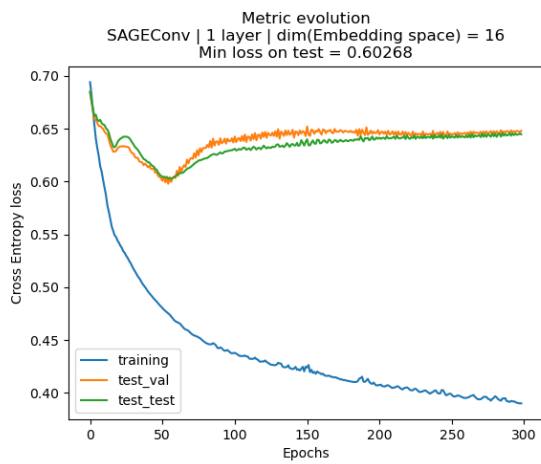


Figure 13: SAGE Conv 1 layer dim 16

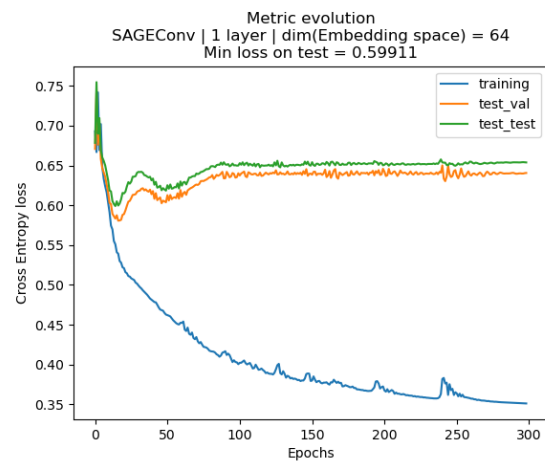


Figure 16: SAGE Conv 1 layer dim 64

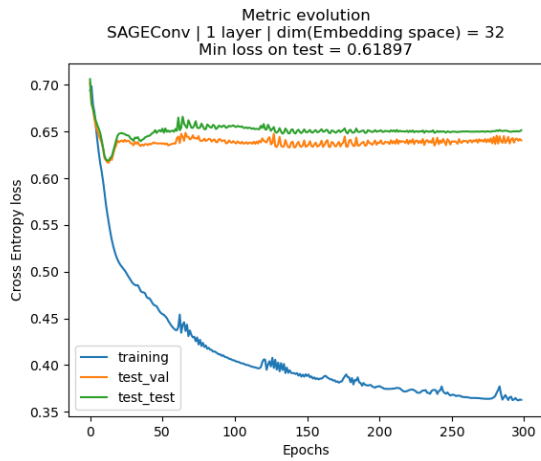


Figure 14: SAGE Conv 1 layer dim 32

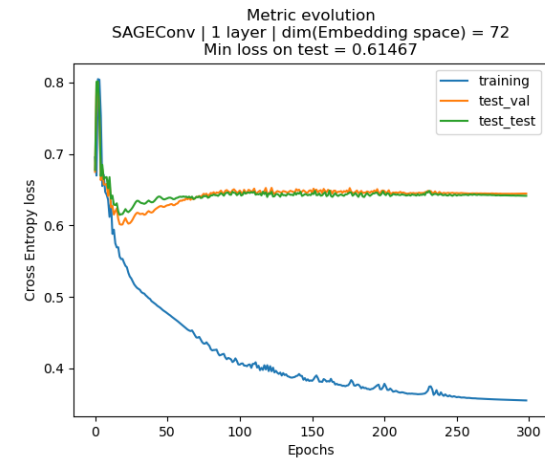


Figure 17: SAGE Conv 1 layer dim 72

Figure 18: SAGE Conv 1 layer dim 96

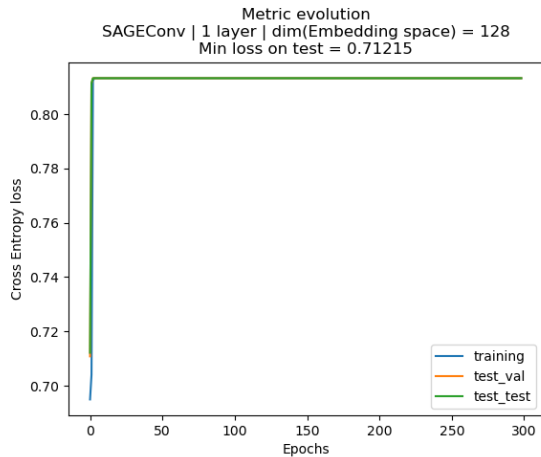


Figure 19: SAGE Conv 1 layer dim 128

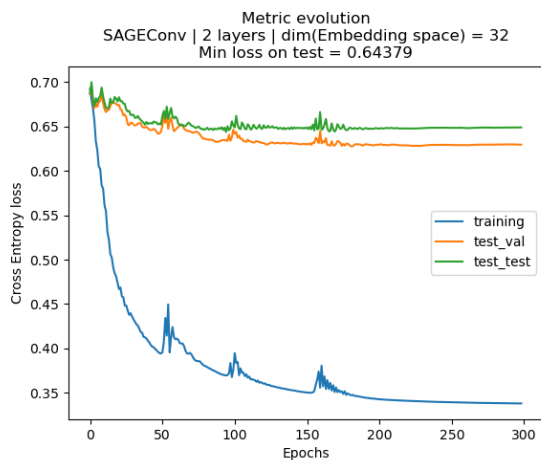


Figure 20: SAGE Conv 2 layer dim 32

### 7.3 Source Code

To improve the reproducibility of our results, the network's source code is available on <https://github.com/mdolr/ter>

### 7.4 Contact

- Dolores Maxime : [maxime.dolores@etu.univ-nantes.fr](mailto:maxime.dolores@etu.univ-nantes.fr)
- Pourtaud Robin : [robin.pourtaud@etu.univ-nantes.fr](mailto:robin.pourtaud@etu.univ-nantes.fr)

## References

- [Sca+09] Franco Scarselli et al. "The Graph Neural Network Model". In: *IEEE Transactions on Neural Networks* 20.1 (2009), pp. 61–80. DOI: 10.1109/TNN.2008.2005605.
- [Cha+14] Kai-Wei Chang et al. "Typed Tensor Decomposition of Knowledge Bases for Relation Extraction". In: *EMNLP*. 2014.
- [Wan+14] Zhen Wang et al. "Knowledge Graph Embedding by Translating on Hyperplanes". In: *AAAI*. 2014.
- [Ji+15] Guoliang Ji et al. "Knowledge Graph Embedding via Dynamic Mapping Matrix". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Beijing, China: Association for Computational Linguistics, July 2015, pp. 687–696. DOI: 10.3115/v1/P15-1067. URL: <https://aclanthology.org/P15-1067>.
- [HYL17] William L. Hamilton, Zitao Ying, and Jure Leskovec. "Inductive Representation Learning on Large Graphs". In: *NIPS*. 2017.
- [KW17] Thomas Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks". In: *ArXiv abs/1609.02907* (2017).
- [Vas+17] Ashish Vaswani et al. "Attention is All you Need". In: *ArXiv abs/1706.03762* (2017).
- [Sch+18] M. Schlichtkrull et al. "Modeling Relational Data with Graph Convolutional Networks". In: *ArXiv abs/1703.06103* (2018).
- [Vel+18] Petar Velickovic et al. "Graph Attention Networks". In: *ArXiv abs/1710.10903* (2018).
- [Yin+18] Rex Ying et al. "Hierarchical Graph Representation Learning with Differentiable Pooling". In: *ArXiv abs/1806.08804* (2018).
- [Cai+19] Ling Cai et al. "TransGCN: Coupling Transformation Assumptions with Graph Convolutional Networks for Link Prediction". In: *Proceedings of the 10th International Conference on Knowledge Capture* (2019).
- [GC19] Liyu Gong and Qiang Cheng. "Exploiting Edge Features for Graph Neural Networks". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2019), pp. 9203–9211.
- [Sun+19] Zhiqing Sun et al. "RotatE: Knowledge Graph Embedding by Relational Rotation in Complex Space". In: *ArXiv abs/1902.10197* (2019).
- [Xu+19] Keyulu Xu et al. "How Powerful are Graph Neural Networks?" In: *ArXiv abs/1810.00826* (2019).
- [Ye+19] Rui Ye et al. "A Vectorized Relational Graph Convolutional Network for Multi-Relational Network Alignment". In: *IJCAI*. 2019.
- [Zha+19] Shuai Zhang et al. "Quaternion Knowledge Graph Embeddings". In: *NeurIPS*. 2019.
- [Vas+20] Shikhar Vashishth et al. "Composition-based Multi-Relational Graph Convolutional Networks". In: *ArXiv abs/1911.03082* (2020).

- [Xie+20] Zhiwen Xie et al. “ReInceptionE: Relation-Aware Inception Network with Joint Local-Global Structural Information for Knowledge Graph Embedding”. In: *ACL*. 2020.
- [Yu+20] Donghan Yu et al. “Generalized Multi-Relational Graph Convolution Network”. In: *ArXiv abs/2006.07331* (2020).
- [Zha+20] Muhan Zhang et al. “Revisiting Graph Neural Networks for Link Prediction”. In: *ArXiv abs/2010.16103* (2020).
- [Cha+21] Haoyu Chang et al. “An Improved TransE Algorithm”. In: *DEStech Transactions on Computer Science and Engineering* (2021).
- [Pro] Vanishing Gradient Problem. *Vanishing gradient problem - Wikiwand*. URL: [https://www.wikiwand.com/en/Vanishing\\_gradient\\_problem](https://www.wikiwand.com/en/Vanishing_gradient_problem).