Deep Learning based Graph embedding for nodes clustering

Florian Le Bronnec
f.le-bronnec@student-cs.fr

António Loison
antonio.loison@student-cs.fr

Abstract

In this project our goals will be to investigate unsupervised learning methods for graph analysis. We will especially focus on the problems of embedding and clustering, with a big emphasis on deep learning approaches.

1. Motivation and project definition

Used in many applications such as community detection in social networks [4], customer group segmentation [5], or genomic feature discovery [2], graph clustering is a very important topic in network science.

In particular, our group aims at studying the subject of graph embedding for clustering. In fact, graphs can be extremely complex objects and simplifying them in order to extract key information is essential. Embedding and clustering are two major techniques that can be applied to achieve this goal.

Clustering Graph clustering is one of the most popular tasks in graph theory. It aims at discovering groups of similar nodes in a graph. This notion of similarity is probably one of the most important of this topic, because its definition can lead to very different kinds of algorithms.

Embedding An embedding is simply another representation of a graph that is as compact as possible, while carrying all the necessary information. To give some example, we often want our embedding to be vectors so that we can use classical machine learning techniques.

Along this project, we are going to see that embedding and clustering are in fact extremely related to each other, through a progressive discovery of the specificities of graphs. Nowadays, deep learning capabilities to find meaningful embeddings are well known, and works extremely well. Therefore it is no surprise that we are going to investigate some of these techniques. Moreover, with the current gain of popularity of deep learning approaches, we also think that it could be challenging to learn more about Graph Neural Network.

The main goal of this project will be to tackle the problem of clustering for graphs, with a special attention to the learned embeddings. We will travel a bit through time by looking at different methods up to the most recent approaches. Our results will be illustrated through two citation datasets, Citeseer and Cora.

2. Proposed approach

Modelling a problem using graphs is a challenging approach. Indeed, a graph is first a structure, with nodes connected by edges. However, each node may contain some information. Classical clustering machine learning algorithms mostly use this information (KMeans for example operates on a euclidean space). On the other side, new approaches have been tried to exploit the structure of the graph. The canonical example is the Spectral Clustering, based on the first eigenvectors of the Laplacian. It seems that, using these techniques, we are a bit trapped between one or the other. But of course, new techniques have emerged to take full advantage of the graph structure and content as well.

3. Problem statement and definition

In this project, we assume that we are given a graph G with N nodes, each nodes having some features, represented by $X \in \mathbb{R}^{N \times p}$ and a set of edges that can be represented by an adjacency matrix $A \in \mathbb{R}^{N \times N}$.

3.1. Content only

3.1.1 KMeans

Methods using only the content just operate classical clustering algorithms on X. We chose to use KMeans as the reference for this kind of methods because it is a method often applied to more complex cases, where the embedding is not trivial. Hence, seeing its performance only on the given embedding can give insights on how meaningful the content is on the nodes for the clustering task.

3.2. Structure only

3.2.1 Spectral Clustering

As said in the introduction of this section, some algorithms operate only on the structure of the graph. But in many cases they amount to finding a new embedding based on the structure of the graph. Then, the final step is ran by classical KMeans clustering. The canonical example we used is the Spectral Clustering. Indeed, we will see just after in Sec. 3.3.1 a generalisation of this algorithm that can take content into account, hence having spectral clustering as a baseline will be useful.

3.3. Structure and content

In the sections described above it may seem that we are a bit trapped between two options: considering only the content or only the structure. But, in fact, more recent approaches have emerged, like RMSC which introduces an interesting formulation of the problem to tackle both these issues at the same time. We will show its formulation because it is a rather elegant one and then introduce new deep learning approaches which, in a way, rely on the same ideas but let the model discover the most convenient representation.

3.3.1 RMSC

RMSC's (Robust Multi-View Spectral Clustering) setup is the following: we assume that we have a set of n nodes, V and *several* weight matrices $W^1, \dots, W^p \in \mathbb{R}^{n \times n}$. We can see that as a direct generalisation of the classical graph modeling problem. Usually, we have only *one* weight matrix, which is the adjacency matrix. Here we simply suppose that we have several "adjacency matrices" that carry different kinds of relationships between the nodes. In practice, we will often have $p = 2$, with W^1 the adjacency matrix and W^2 a similarity matrix on the features of each node. Let's go back to the more general setup, with $p \geq 2$. An efficient way to model the relationships within a given graph is to look at its transition matrix. We are hence going to introduce the transition matrices $P^i = (D^i)^{-1}W^i$, where D^i is the diagonal degree matrix (the row-wise sum of W^i). The RMSC is based on some major assumptions that are crucial for a deep understanding of the method.

Assumption 1. *Each W^i is by itself sufficient to retrieve most of the clustering information.*

This assumption allows us to postulate the existence of a ground truth transition matrix $\widehat{P}$, such that:

$$\forall i \in [1..p],\ P^i = \widehat{P} + E^i. \tag{1}$$

Equation (1) can be further investigated in order to gain some insights on what $\widehat{P}$ and E^i look like.

Assumption 2. *$\widehat{P}$ is low rank.*

Indeed, assume on a toy example that the labels of our nodes are $(0, 0, 0, 1, 1, 2, 2,)$ and that the classes are extremely well connected together, with almost no connection outside of the class. Then the corresponding transition matrix will be block diagonal:

$$\begin{pmatrix} 1/3 & 1/3 & 1/3 & 0 & 0 & 0 & 0 \\ 1/3 & 1/3 & 1/3 & 0 & 0 & 0 & 0 \\ 1/3 & 1/3 & 1/3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 0 & 0 & 0 & 1/2 & 1/2 \end{pmatrix} \tag{2}$$

Of course, the rank of the matrix will be preserved if we swap the columns and rows and will mostly be preserved if we assume that the interactions between classes is negligible.

Assumption 3. *E^i is of small magnitude and is sparse.*

Indeed, this can be seen as a consequence of Assumption 1, as we should be able to retrieve most of the information from the different views, the transition matrices should not differ too much from $\widehat{P}$. And we assume that some of the coefficients are completely unaltered, meaning that E^i should have some sparsity properties.

With all these assumptions, solving Eq. (1) can be formulated as an optimization problem under constraint:

$$\begin{aligned} &\min_{\widehat{P},\,(E^i)_i} \operatorname{rank} \widehat{P} + \lambda \sum_{i=1}^{p} \left\| E^i \right\|_1, \\ &\text{such that, } \begin{cases} \forall i \in [1..p],\ P^i = \widehat{P} + E^i, \\ \widehat{P} \geq 0,\ \widehat{P}\mathbf{1} = \mathbf{1}. \end{cases} \end{aligned} \tag{3}$$

Remark 1. *The sparsity prior may seem a bit surprising: as we want to think of E^i as noise, we could have expected a L_2 norm which is only going to penalize the magnitude of the noise. But we should keep in mind that we are working with a transition matrix. In fact it is more accurate to say that it is the weight matrices W^i that are noisy. But their transition matrices should remain globally unaffected if the noise is of small amplitude. Hence, we can expect that only some coefficients of the transition matrices will be affected, which justifies the sparsity prior.*

As usual, we can make this problem convex by replacing the rank by a nuclear norm (L_1 norm on the singular values vectors of a matrix):

$$\min_{\widehat{P},\,(E^i)_i} \left\|\widehat{P}\right\|_* + \lambda \sum_{i=1}^{p} \left\|E^i\right\|_1, \quad \text{such that, } \begin{cases} \forall i \in [1..p],\ P^i = \widehat{P} + E^i, \\ \widehat{P} \geq 0,\ \widehat{P}\mathbf{1} = \mathbf{1}. \end{cases} \tag{4}$$

The convex optimization problem (4) can be solved iteratively, by introducing the augmented Lagrangian of the problem, with an auxiliary variable Q. The algorithm is described in Algorithm 1.

Once this transition matrix is built, we can use, as in the random walk setting, a formulation equivalent to spectral clustering. This is described in Algorithm 2. The final pipeline is described in Algorithm 3.

Algorithm 1: RMSC: transition matrix

Data: $P^1, \ldots, P^p$ the observed transition matrices, $\lambda > 0$ the regularization parameter, M the maximum number of iterations, ε the stopping criterion.

Result: $\widehat{P}$, the ground truth transition matrix

Initialization: $\widehat{P} = 0$, $Q = 0$, $Z = 0$, $\mu = 10^{-6}$, $\rho = 1.9$, $\mu_{max} = 10^{10}$.

for $i \in [0..M]$ **do**

$\widehat{P} \leftarrow \frac{1}{p+1}\left(Q + \frac{1}{\mu}Z + \sum_{i=1}^{p}\left[P^i - E^i - \frac{1}{\mu}Y^i\right]\right)$

Project each column of $\widetilde{P}$ onto the probability simplex and store the result in $\widehat{P}$.

$$\forall i \in [1..p],\ E^i = \mathrm{prox}_{\lambda/\mu}\left(P^i - \widehat{P} - \frac{1}{\mu}Y^i\right). \tag{5}$$

$U, \Sigma, V \leftarrow \mathrm{SVD}\left(\widehat{P} + \frac{1}{\mu}Z\right)$,

$Q \leftarrow U\,\mathrm{prox}_{1/\mu}(\Sigma)V^T$,

$Z \leftarrow$

$\forall i \in [1..p],\ Y^i \leftarrow Y^i + \mu\left(\widehat{P} + E^i - P^i\right)$

$\mu \leftarrow \min(\mu, \mu_{max})$.

if *the norm of the update of* $\widehat{P}$ *is smaller than* ε **then**

Return $\widehat{P}$.

end

end

Return: $\widehat{P}$.

Algorithm 2: RMSC: MC spectral clustering

Data: $\widehat{P} \in \mathbb{R}^{n\times n}$ the transition matrix, K the number of clusters.

Result: $U \in \mathbb{R}^{n\times K}$ the embedding of the nodes

Find the eigenvector π of $\widehat{P}^T$ associated to the eigenvalue $\lambda_\pi = 1$.

Compute the Laplacian matrix:

$$L = \mathrm{diag}(\pi) - \frac{1}{2}\left(\mathrm{diag}(\pi)\widehat{P}^T + \widehat{P}\,\mathrm{diag}(\pi)\right). \tag{6}$$

Return: $U \in \mathbb{R}^{n\times K}$ the matrix of the K eigenvectors corresponding to the K smallest eigenvalues.

Algorithm 3: RMSC

Data: $P^1, \ldots, P^p$ the observed transitions matrices, $\lambda > 0$ the regularization parameter, M the maximum number of iterations, ε the stopping criterion.

Result: $Y \in [0..K-1]^n$ the labels of the nodes.

Get $\widehat{P}$ by applying Algorithm 1.

Get the embedding U by applying Algorithm 2.

Apply KMeans on U.

Return: the labels yielded by KMeans.

3.4. Variational autoencoder

These approaches are extremely appealing because they often provide a way to use both graph's structure and content. An approach using deterministic autoencoder is presented later in this project in Sec. 3.5.2.

We will present here a baseline approach which consists of a Variational Autoencoder and finally the new contribution of [8].

3.4.1 VAE

A Variational Autoencoder (VAE) can be seen as a stochastic approach of the classical Autoencoder. Indeed, rather than mapping a point to a deterministic low dimensional embedding, a VAE maps it to a probability distribution. The advantage of this model is that, during the training, the nodes do not have a fixed representation that could be sometimes too restrictive. And therefore, during the training, the network will see several representations for the same points which on average carry the same information. This stochasticity that allows the model to explore different parts of the

latent space will be much appreciated during the training (and can also give some interpretability afterwards).

We suppose that our model depends on a latent random variable $Z \in \mathbb{R}^l$, with l the latent space dimension.

Encoding To *encode* our graph data (X, A) into a probability distribution, we use a parametrized model, as described in [6]:

$$q(Z \mid A, X) = \prod_{i=1}^{n} \mathcal{N}(z_i \mid \mu_i, \mathrm{diag}(\sigma_i^2)), \tag{7}$$

where $\mu_i \in \mathbb{R}^l$ and $\sigma_i^2 \in \mathbb{R}^l$ are parametrized by graph neural networks. Indeed, in this setup, we simply assume that the encoding of a node x_i depends on the structure of the graph and on its features. Hence, we see that the content and structure are naturally embedded together. For this graph we used two implementations. The first one is based on Graph Convolutional Layer and the second one on Graph Attention Layer (which can be seen as a generalisation of the former).

Encoder Architectures Let $\boldsymbol{\mu}, \boldsymbol{\sigma} \in \mathbb{R}^{n \times l}$ be the matrices of the means and diagonal covariances of each row of Z. Then, we introduce a first layer h_0 (without any activation function applied to the output) and $h_{\boldsymbol{\mu}}, h_{\boldsymbol{\sigma}}$ such that:

$$\begin{cases} \boldsymbol{\mu} = h_{\boldsymbol{\mu}}(\mathrm{ReLU}(h_0(X, A)), A), \\ \boldsymbol{\sigma} = h_{\boldsymbol{\sigma}}(\mathrm{ReLU}(h_0(X, A)), A), \end{cases} \tag{8}$$

with h being either a graph convolutional layer or a graph attention layer.

Decoder We will train our VAE to reconstruct the graph given the features (as in practice we often have access to the features but not to the relationships between nodes). Hence, we will naturally aim at reconstructing the adjacency matrix A. This reconstruction will still be stochastic, hence we naturally assume that $p(A_{ij} \mid z_i, z_j) \sim \mathcal{B}(\mathrm{Sigmoid}(z_i^T z_j))$.

Training The training of a VAE aims at maximizing the log-likelihood of the model:

$$l = \log p(A), \tag{9}$$

where:

$$p(A) = \int p(A \mid Z) p(Z) \, \mathrm{d}Z. \tag{10}$$

As the integral is untractable, we will estimate it using Monte Carlo methods. And because we will be interested in reducing the variance of such an estimate, we compute it thanks to importance sampling, using a parametrized distribution, $q(z \mid X, A)$.

And it can be shown with rather simple arguments that:

$$\mathcal{L}_K = \mathbb{E}_{Z^1, \dots, Z^K \sim q(Z \mid X, A)} \left[\log \frac{1}{K} \sum_{k=1}^{K} \frac{p(A \mid Z^k)}{q(Z^k \mid X, A)} \right] \tag{11}$$

is in fact a lower bound of the log-likelihood. And using the reparametrization trick (i.e optimizing on the parameters of the distributions by applying a function to sample from a simpler distribution), this lower bound is quite simple to maximize using automatic differentiation. This lower bound is called the IWAE estimate, from the great article [1].

Remark 2. *Using $K = 1$, one retrieves the original ELBO bound used in the article [6]. And using Jensen's inequality we can also show that $\mathcal{L}_1 \leq \mathcal{L}_2 \leq \cdots \leq \mathcal{L}_K$, which shows that $\mathcal{L}_K$ is a tighter lower bound than the ELBO bound. For this reason, we will call this model VAE when $K = 1$ and IWAE-K when $K > 1$.*

Remark 3. *In practice, A will often be very sparse, as it is an adjacency matrix. Hence in order to get a decent result, we will have to re-weight $p(A_{ij} \mid Z^k)$ in the reconstruction part of the loss.*

Remark 4. *In the expression of $\mathcal{L}_K$ there is an expectation. We are obviously not going to optimize directly $\mathcal{L}_K$ as it is intractable but simply an unbiased estimate of $\mathcal{L}_K$ by sampling $Z^1, \dots, Z^K$ from the distribution $q(Z \mid X, A)$.*

Embedding The final task of getting an embedding is simply achieved by sampling for each (X_i, A_i) say 1000 $Z^i \sim q(Z \mid X, A)$ whose parameters have been learned, and then compute the mean. Then KMeans can be ran on this embedding.

3.4.2 VAE refinements

A drawback of the VAE approach is that it is mainly optimized for reconstruction purpose, especially through the choice of the prior being a standard gaussian. And maximizing the lower bound $\mathcal{L}_K$ implicitly considers a regularization that forces the output to be collectively similar to the prior (KL regularization in the case $K = 1$).

Hence instead of using a gaussian prior, we used a mixture of gaussian, with standard convariance matrices and randomly initialized means. Indeed, we can expect that with such a prior the clusters will be more spread out in the latent space.

3.5. Joint optimization

Until now we have just been learning powerful encodings. However, our main goal is to achieve clustering. Therefore, we can wonder if there is a way to optimize the encoding so that it takes into account this clustering task.

3.5.1 VAE-EM

The first idea we got was to leverage the new prior we used in Sec. 3.4.2 and optimize the prior jointly with the encoding. Indeed, remember that in VAE framework we only use the prior's density but do not perform any optimization on it. And because we are now using a GMM prior, we had the idea to optimize it using the EM algorithm. This very simple idea can be summarized as follows:

1. Optimize the VAE by running $n_{initial}$ epochs (just like we did in before) in order to learn a first encoding.
2. For the remaining epochs, keep optimizing the VAE but every n_{prior} epochs do:
   (a) Sample K_{EM} data points from the conditional density $p(Z|X, A)$.
   (b) Optimize the GMM prior by running n_{EM} EM iterations using these K_{EM} data points.

Here we fully leveraged the stochastic approach of VAEs.

3.5.2 Attributed Graph Clustering with a Deep Attentional Embedding

This approach combines both clustering and embedding approaches in a unified framework. First the embeddings are pre-learned using a graph attention neural network and then the network and the centroids of the clusters are jointly optimized to achieve the best results.

Using deep learning for graph embedding and clustering is something known to work well in practice. But they often are used one after the other (e.g deep learning embedding followed by KMeans or spectral clustering) and thus do not benefit from all the synergies they can have together. This is why we propose to study the approach presented in [8] which combines all these techniques in a unified framework, called *Deep Attentional Embedded Graph Clustering*. This fundamental difference between these two approaches is summarized in Figure 1, extracted from the cited paper:

This architecture first uses an auto-encoder and then performs joint optimization to become more specific to clustering purposes.

First, the autoencoder is pretrained to learn a vector representation of the nodes, taking into account the features of each node and their structural context.

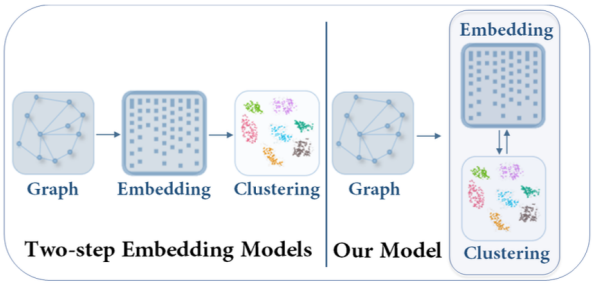

Figure 1. The difference between two-step embedding learning models and the proposed model from [8]

Encoder The encoder is built with 2 attention layers [7]. Each attention layer l takes as input the current node embedding for each node $z^{(l)}$ and returns a new embedding $z^{(l+1)}$, such that:

$$z_i^{(l+1)} = \sigma\left(\sum_{j \in N_i} \alpha_{i,j} W z_j^{(l)}\right) \tag{12}$$

$z_j^{(l)}$ denotes the input representation of node j at layer l. W is a parametrized matrix, N_i is the set of neighbors of node i and $\alpha_{i,j}$ is the attention coefficient such that:

$$\alpha_{i,j} = \text{Softmax}_{r \in N_i}(c_{i,j}) \tag{13}$$

with

$$c_{i,j} = \text{LeakyReLU}(M_{i,j} \overrightarrow{a}^T [Wx_i \,||\, Wx_j]) \tag{14}$$

Here, $M_{i,j}$ is the coefficient of the matrix $M = (D^{-1} + D^{-2})/2$, $\vec{a} \in \mathbb{R}^{2 \times \text{number of output features}}$, and $||$ is the concatenation operator and D is the degree matrix.

Remark 5. *This is a classical approach of graph encoding, the deterministic version of VAEs. The attentive reader may note that we are using an architecture similar to graph attention, but with the addition of the $M_{i,j}$ coefficients.*

Decoder As in the previous architecture, we simply retrieve the adjacency matrix with a sigmoid on the dot product with the node's embeddings.

$$\hat{A}_{i,j} = \sigma(z_i^T z_j) \tag{15}$$

The final loss is computed with a binary cross entropy loss between the predicted adjacency coefficients and the true adjacency matrix.

$$L_r = \sum_{i,j} \text{BCE}(\hat{A}_{i,j}, A_{i,j}) \tag{16}$$

Joint optimization Once we have the pre-trained embeddings, we can find a first set of clusters using K-Means. Then we will refine the embeddings and the cluster centroids using a self-optimization objective. The problem is that we do not have cluster labels towards which we should adapt our parameters. Therefore, we will compute two distributions P and Q giving the probability of each node to belong to each cluster. The Q distribution is computed in order to have soft labels, while the P distribution gives stronger labels that will be used as ground truth.

The Q distribution derives from the Student's t-distribution with one degree of freedom as follows:

$$q_{i,u} = \frac{(1 + \|z_i - \mu_u\|)^{-1}}{\sum_k (1 + \|z_i - \mu_k\|)^{-1}}. \tag{17}$$

This idea directly comes from the t-SNE dimensionnality reduction algorithm.

The P distribution is then given by:

$$p_{i,u} = \frac{q_{i,u}^2 / \sum_i q_{i,u}}{\sum_k q_{i,k}^2 / \sum_i q_{i,k}}. \tag{18}$$

These expressions seem a bit original, but Remarks 6 and 7 provide some additional details.

Remark 6 (Q expression)**.** *Q should only be seen as a soft assignment of the samples given the cluster centers. A Cauchy-like prior is used on Q because it appears in previous work that this similarity measure behaves well (for example it is not dependent on the scale of the clusters).*

Remark 7 (P expression)**.** *For P, it is a bit more audacious. Indeed, as we square $q_{i,u}$, the soft assignments where the algorithm is more confident will be increased while the ones where it is lower will be decreased. Hence, using a* KL *loss will likewise force Q to be more confident on the assignments. However, by doing this, it implicitly assumes that the initial encoding contains features good enough to make a reliable soft assignment.*

Final loss The joint objective is then to minimize the Kullback-Leibler divergence between P and Q while keeping L_r also low. So, the final loss is given by:

$$\begin{aligned} L &= L_r + \gamma \,\mathrm{KL}(P \,||\, Q) \\ &= L_r + \gamma L_c, \end{aligned}$$

where gamma is a hyperparameter that multiplies L_c to have around the same importance as L_r.

Training We train the entire framework to minimize L. In Fig. 2, it is visible that it is mostly the KL loss that is optimized. But the update of the P distribution does not allow it to converge to a minimum. Nevertheless, empirically, we see that the metrics are well improved by this joint optimization step.

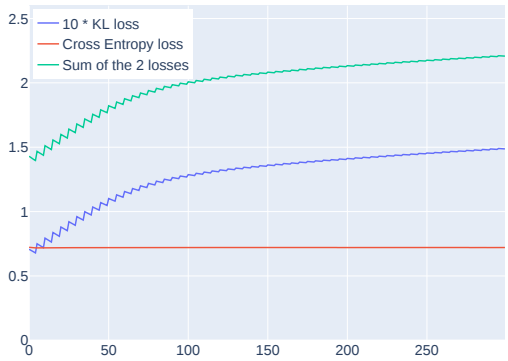

Figure 2. Evolution of the 3 losses.

Then, the labels s_i of each node are obtained with:

$$s_i = \underset{u}{\operatorname{argmax}}\, q_{i,u}. \tag{19}$$

However, to avoid instabilities in the optimization process, the P distribution is updated only every 5 epochs, as with what we did for the VAE-EM. Otherwise it becomes a bit difficult for the network to fit a fast moving objective.

Remark 8 (Multi Head Attention)**.** *We tried to train a multi head attentional auto-encoder. But it did not give better results than the single head autoencoder.*

4. Implementation

A big part of our work was spent on the implementation of each algorithm.

KMeans & Spectral Clustering For KMeans we simply used Scikit-Learn implementation. For Spectral Clustering it was straightforward to implement from scratch.

RMSC RMSC was implemented from scratch. The main difficulties was the projection of a matrix onto the probability simplex. No Python implementation existed for this purpose. We hence implemented it in a vectorized way in Numpy. The bottleneck of the algorithm is the SVD calculation (which we need to do on the whole Laplacian matrix).

VAE & IWAE We wanted to rely as much as possible on the DGL library [9]. However no implementation was available for graph variational autoencoder. And more generally, the only existing implementations found on the internet were often in Tensorflow and they were all limited to a standard gaussian prior. And they were not implementing the IWAE variant either. We therefore propose a brand

new implementation based on the efficient Pytorch distribution library [3] that makes it possible to use a lot of other distribution as prior or as instrumental densities and which totally exploits the strength of the DGL library.

DAEGC DAEGC implementation differs from classical graph architecture that are implemented out of the box in DGL. This is the reason why this model does not rely on the DGL module. However we believe implementing these modified GATConv layers using the M matrix is something one might want to think of for the future. The proposed implementation was also coded from scratch taking inspiration on the official code associated with [8].

5. Results

The common aspect of the algorithms we tried is that they all yield a graph encoding. It can then be useful to visualize it to get insights on the algorithm performances.

5.1. Content or structure only

Figures 3 and 4 present the result of KMeans and spectral clustering. We clearly see the downsides of considering only one aspect of the problem. Either the structure is not visible at all in the clustering, or its strength is too high.

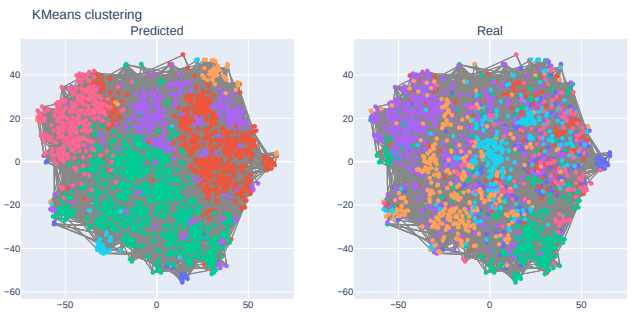

Figure 3. We see that the structure of the graph does not appear at all on this clustering.

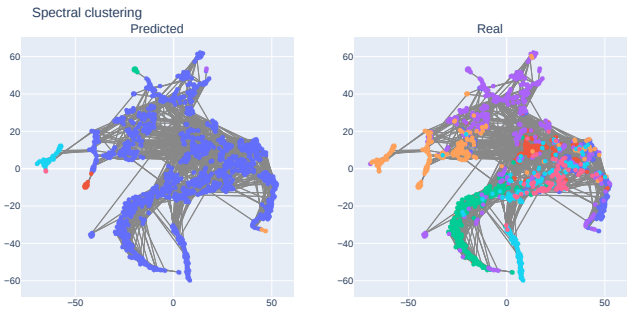

Figure 4. On the contrary, here the embedding depends too much on the structure and hence is not accurate at all, due to a lot of outliers.

5.2. Content and structure

Let's now visualize the result of RMSC embedding on Fig. 5. The gain is huge compared to previous approaches, and we can notice a lot of similarity with the ground truth labels. We can even be surprised by the quality of the result for a non-deep approach.

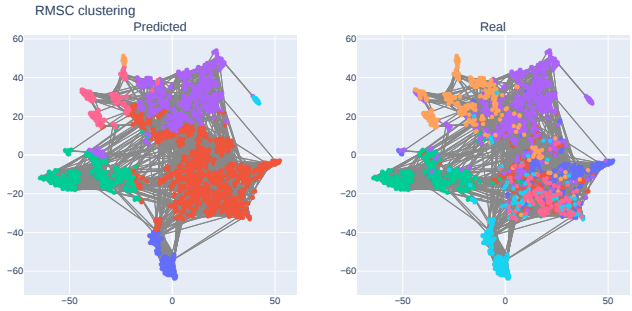

Figure 5. The clusters are well delimited and the underlying structure is also quite visible. We observe a lot of similarity between the ground truth and the predicted assignments.

And for an IWAE embedding, see Fig. 7.

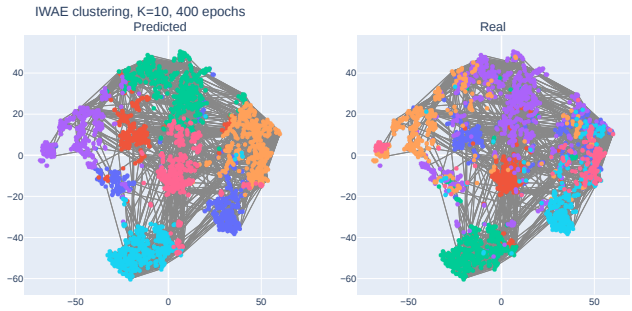

Figure 6. Here again we observe that the clusters seem quite close to the ground truth labelling.

For the difference between VAE and IWAE, it is especially visible during the training, see Fig. 8.

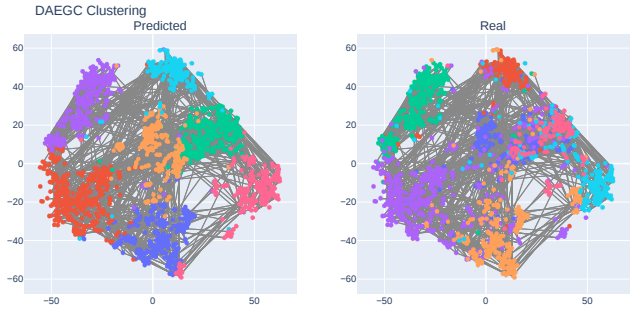

Figure 7. With DAEGC, the learned embeddings allow us to have good cluster separation

Remark 9 (VAE refinements)**.** *Our two refinements lead to different improvements. The change of prior indeed made a small increase in the performances of the VAEs, therefore we kept it. But the EM steps had a negligible effect. This is the reason why VAE-EM does not appear in the results. We still wanted to mention it because it is a good introduction to DAEGC and this is not something commonly done with VAEs.*

Figure 8. Citeseer dataset. IWAE converges a lot faster than VAE and achieves an overall better maximum.

Method	COMP	HOM	NMI	ARI
KMeans	0.243	0.190	0.214	0.113
SC	0.113	0.016	0.028	-0.009
RMSC	0.435	0.350	0.388	0.199
VAE	0.434	0.451	0.443	0.369
IWAE	0.457	0.481	0.470	0.387
Pre-DAEGC	0.515	0.532	0.523	0.483
DAEGC	**0.531**	**0.544**	**0.538**	**0.506**

Table 1. Comparison between our methods on the Cora Dataset.

Method	COMP	HOM	NMI	ARI
Kmeans	0.221	0.220	0.220	194
SC	0.029	0.015	0.020	0.009
RMSC	**0.449**	0.275	0.341	0.203
VAE	0.298	0.294	0.296	0.281
IWAE	0.324	0.318	0.321	0.306
Pre-DAEGC	0.412	0.41	0.411	0.412
DAEGC	0.424	**0.427**	**0.425**	**0.450**

Table 2. Comparison between our methods on the Citeseer Dataset.

Looking at the results, it is clear that algorithms that consider both the structure and the content are much more efficient. The DAEGC method are by far the most effective. In particular, Pre-DAEGC corresponds to the results of DAEGC before the joint optimization step. We see that this step clearly improves the results. Yet, we want to acknowledge the surprisingly good results of RMSC clustering, which outperforms VAE on some metrics, and which yields great visualization results. It highlights that since deep learning can be extremely powerful, we should never neglect the importance of good modelisation and problem formulation that can sometimes give very impressive results.

5.3. Metrics explanation

We are dealing with unsupervised methods but we have access to the ground truth labels, corresponding to real communities in our graphs. We are going to describe how we can measure the quality of our clustering using this ground truth.

Homogeneity quantifies if each cluster contains member of a single class. 1 is best and 0 is worst.

Completeness quantifies if all samples of a given cluster are assigned to the same cluster. 1 is best and 0 is worst.

Normalized mutual information quantifies whether the two assignments are dependent. 1 means perfect correlation and 0 means no obvious correlation.

Adjusted rand index measures the similarity between two assignments. Random assignments have an ARI close to 0 and same labellings have an ARI of 1.

6. Conclusion

During this project we have conducted a heavy review of the different clustering methods that have been tried with graphs and also proposed some new ones (IWAE for graphs and VAE-EM), with a big emphasis on implementation. We showed how using structure and content can give much better results, but at a cost of a more advanced formulation. Indeed, RMSC clustering has been published in 2014, which is quite new compared to KMeans or Spectral Clustering, and Variational Autoencoder have been first used in 2016. And now new state of the art techniques are exploiting the potential of deep learning approaches, which can be optimized for clustering, but one should keep in mind that efficiently designed non-deep methods can still give extremely satisfying and more interpretable results.

References

[1] Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders, 2015. 4

[2] Irineo Cabreros, Emmanuel Abbe, and Aristotelis Tsirigos. Detecting community structures in hi-c genomic data, 2015. 1

[3] Pytorch community. Pytorch distribution. 7

[4] Pengwei Hu, Keith C.C. Chan, and Tiantian He. Deep graph clustering in social network. In *Proceedings of the 26th International Conference on World Wide Web Companion*, WWW '17 Companion, page 1425–1426, Republic and Canton of Geneva, CHE, 2017. International World Wide Web Conferences Steering Committee. 1

[5] Su-Yeon Kim, Tae-Soo Jung, Eui-Ho Suh, and Hyun-Seok Hwang. Customer segmentation and strategy development based on customer lifetime value: A case study. *Expert Systems with Applications*, 31(1):101–107, 2006. 1

[6] Thomas N. Kipf and Max Welling. Variational graph auto-encoders, 2016. 4

[7] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017. 5

[8] Chun Wang, Shirui Pan, Ruiqi Hu, Guodong Long, Jing Jiang, and Chengqi Zhang. Attributed graph clustering: A deep attentional embedding approach, 2019. 3, 5, 7

[9] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019. 6