Task 4

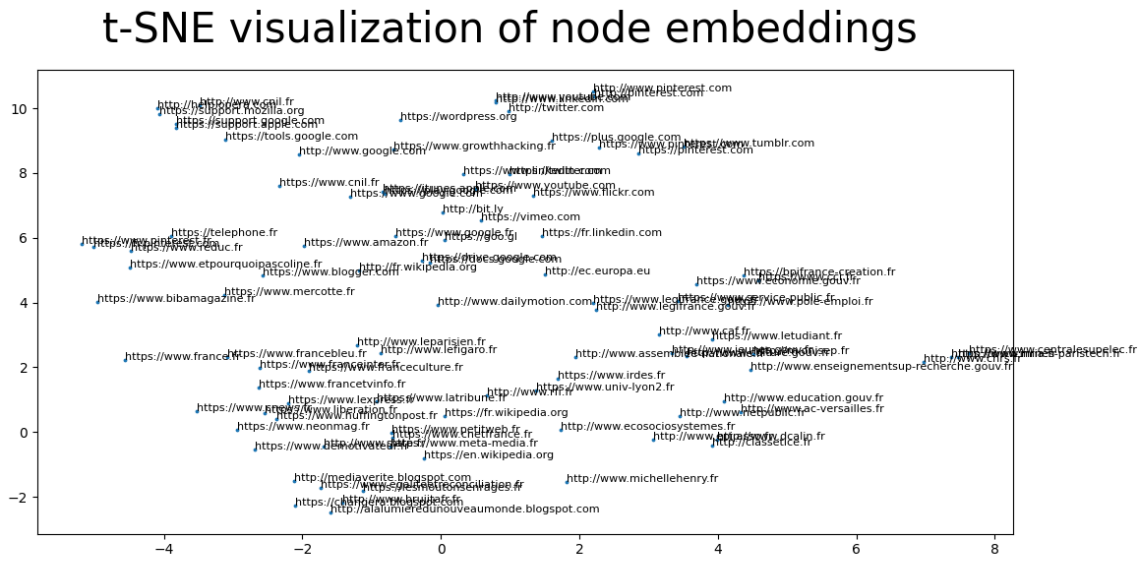

Figure 1: Visualisation of the 128-dimensional features learnt by the Deepwalk algorithm for a fraction of the samples (n=100) in a 2-dimensional space using T-SNE. The embeddings look consistent as education.gouv and ac-versailles.fr and univ-lyon2.fr appear close.

Question 1

Deepwalk and Skipgram reminder

Recall that the cosine similarity of two vectors A, B of the same inner product space is defined as

$$S_C(A, B) = \frac{\langle A, B \rangle}{\|A\| \cdot \|B\|}. \tag{1}$$

The embedding of a node v_i using the Deepwalk algorithm is $\phi(v_i)$ is given by

$$\arg\min_{\phi} -\log \prod_{\substack{j=i-w \\ j \neq i}}^{i+w} P(v_j|\phi(v_i)) = \arg\max_{\phi} \sum_{\substack{j=i-w \\ j \neq i}}^{i+w} \log\Big(P(v_j|\phi(v_i))\Big), \tag{2}$$

with $\phi : V \to \mathbb{R}^d$ being the embedding function that maps vertices to their embeddings and w is the size of the sliding window according to the **Skipgram** model.
The set of nodes in the neighborhood, that is visited at a distance w, of a node is called **Corpus**. Deepwalk maximises the likelihood of observing the w previous and the w following vertices visited in the various random walks.
In general, ϕ is encoded in the first layer of a two-layer neural network (W_{in} in Figure 2) with $|V|$ each having a weight dimension equal to d, which gives a $d \times |V|$ model to embed the graph, where $|V|$ is called the size of the vocabulary and d is the embedding dimension.

Graph with M components being K_2 graphs

A random walk in the graph G made of M complete graphs K_2 can only go through the two nodes of the connected component of the starting node. Consider a connected component K and let v_0, v_1 be the two nodes of K.

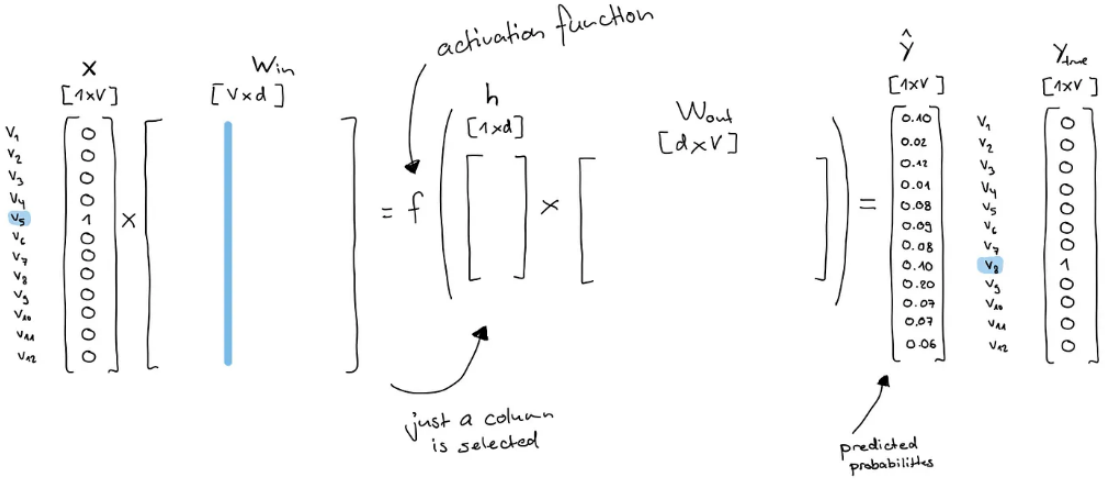

Figure 2: Skipgram for graphs, from [1]. The input is an one-hot encoded vector with length equal to the length of the vocabulary V, it is multiplied with an input-weightmatrix W_{in}, yielding an embedding h. h is then multiplied with a second weightmatrix, W_{out}. The product is passed into an activation function, which is the softmax because we want to model a probability distribution over the output words. We compute the probability for each word in the vocabulary to find it in the corpus of the input node and train the network in a supervised manner, using as labels the true corpus of the input node.

One should remark that, for all $j \notin \{0, 1\}$, v_j cannot be neighbor of v_0 or v_1 in a random walk and therefore cannot appear in the objective function (2) for v_0 (resp. v_1) which only has terms of the type $\log\Big(P(v_1|\phi(v_0))\Big)$ and $\log\Big(P(v_0|\phi(v_0))\Big)$ (resp. $\log\Big(P(v_1|\phi(v_1))\Big)$ and $\log\Big(P(v_0|\phi(v_1))\Big)$). Yet, DeepWalk maximizes the probability of its neighbors in the walk. Thus, we should have

$$P(v_0|\phi(v_1)), P(v_1|\phi(v_1)) > 0 \text{ and } \forall j \notin \{0, 1\}, P(v_j|\phi(v_1)) = 0, \tag{3}$$

and similarly for $P(\cdot|\phi(v_0)$.
Since an embedding in a vector space $\mathbb{R}^d$ of a graph G must assign high cosine similarities to nodes when one has high probability to be visited when starting a random walk from the other, we expect $S_C(v_0, v_1) \simeq 1$ and $S_C(v_0, v_j) \simeq 0$ for all $j \neq 1$.

Task 8

```
Number of nodes: 34
Number of edges: 78
Generating walks
Training word2vec
DeepWalk_y_pred : [0 1 1 0 0 0 1]
Test accuracy: 1.0
 Train accuracy: 1.0
Spectral_y_pred : [0 1 1 1 0 0 1]
Test accuracy: 0.8571428571428571
 Train accuracy: 0.7037037037037037
```

We can see that the DeepWalk embeddings allow for much better test accuracy (1.0) than spectral embeddings (0.857) when evaluated on the karate dataset classification task.

Question 2

- w is the window size.
- d is the hidden dimension in the Skipgram model.
- L is the length of the walks.
- n_{rw} is the number of walks per node.

DeepWalk

Building training random walks

Building the n_{rw} random walks of length L for each node costs $\propto L\gamma|V| = O(|V|)$.

Skipgram

According to Section 3.2 of the paper introducing Word2Vec [3], the time complexity of the algorithm is $\propto C(D + D\log_2(|V|))$, which with our notation is equal to $2w(d + d\log_2(|V|))$, the $\log_2(|V|)$ coming from the exploration of the binary tree of size the vocabulary V (hierarchical softmax).
This is done for each of the e epochs and, for each epoch, we process $n_{rw}|V|$ sentences, so the complexity of the Skipgram algorithm is

$$O\left(en_{rw}|V|2w(d + d\log_2(|V|))\right) \tag{4}$$

which scales in $O(|V|\log(|V|)$ with the size of the graph.

Global complexity

If we sum the complexities of these two steps, we see that the dominating term is the Skipgram complexity, which is $O(|V|\log(|V|)$.

Spectral embedding

According to [4], the time complexity of computing the eigenvectors and eigenvalues of the normalized Laplacian matrix is in general $O(kn^2)$ with k being the dimension of the embeddings we want. Indeed, it is well-known that computing the eigenvectors and eigenvalues of a $n \times n$ matrix has time complexity $O(n^3)$ but here we just compute the k smallest eigenvectors.
Computing the Laplacian costs in general $O(n^2)$ time.
Yet, if the Laplacian is sparse with $O(n)$ non-zero entries, the cost of computing the $n \times k$ with $k << n$ matrix with selected eigenvectors is $O(n)$.

Question 3

Self-loops

$$\hat{A} = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2} \tag{5}$$

As explained in Section 5.1.4 of [2], adding self-loops to the adjacency matrix allows to omit the explicit update step, by setting the hidden state of layer t of the GNN for node u as

$$h^{(t)} = f\left(\{h_v^{(t-1)}, \forall v \in \mathcal{N}(u) \cup \{u\}\}\right), \tag{6}$$

where the aggregation is taken over the set $\mathcal{N}(u) \cup \{u\}$ of the neighbors of u **and** u itself instead of only $\mathcal{N}(u)$, and f is the activation function. If we did not add self-loops and performed aggregation over the neighbors, we would have that the node's hidden state is only updated as a function of the features of its neighbors, without taking into account its own features.
The renormalization trick by the inverse square root of the degree matrix is made to avoid nodes having high degree to have much more importance in the message passing than nodes with lower degree.

Single layer case

On a cycle graph, assume the layers are just doing identity and that the features of the nodes are initialised as being equal.
In the single-layer case, there is only one message passing, so, for each node, the feature is updated using only the information of its neighbors, followed by the linear and softmax layers. Thus, the GNN is asked to learn how to classify the graph's nodes using only the information of the direct neighbors.
Not using the current node's feature for its own update can be fatal to the classification task. To illustrate, let us look at Task 11's situation. We encode the nodes as one hot-encoded vectors of size $|V|$, i.e. each node's input feature for the training is its index. The only linear layer, if it has self-loops, can memorize each training node's label and reach very high training accuracy. But without the self-loops, it is impossible to do such memorisation.

Two layer case

In this case, each node retrieves the information of its input feature thanks to the second layer. Indeed, consider a node u, then the hidden states at the exit of the first GNN layer $\{h_v^{(0)}, \forall v \in \mathcal{N}(u)\}$ depend on its input feature $h_u^{(input)}$. Then, the second message passing layer allows u's second hidden state $h_u^{(1)}$ to depend on $h_u^{(input)}$ through its dependence on $\{h_v^{(0)}, \forall v \in \mathcal{N}(u)\}$. Even if less bad than in the single layer case, it is very indirect and suboptimal, making the classification task difficult.

Task 11

```
Epoch: 100 loss_train: 0.0188 acc_train: 1.0000 time: 0.0015s
Optimization Finished!
Total time elapsed: 0.3109s

Test set results: loss= 0.0019 accuracy= 1.0000
```

Task 12

```
Epoch: 100 loss_train: 0.6731 acc_train: 0.5556 time: 0.0010s
Optimization Finished!
Total time elapsed: 0.2480s

Test set results: loss= 0.7828 accuracy= 0.2857
```

Remark : we get the exact same accuracy on the test test, if we take the embedding matrix to be of dimension $n \times n$ or $n \times 1$, it is only the fact that all the nodes have equal training features that matters.

In this case, we train the GNN to predict the set node's class with all nodes having same feature vector. Thus, the only information that is used in each forward pass for an input is the structural information encoded in $\hat{A}$ that is broadcasted during the two message passing steps of the layers of the GNN.
The task is almost impossible except if each cluster of the graph has a very different structure from the other cluster. For example, the GNN might have had better accuracy (than the current 0.2857 on the test set) if it the two clusters were respectively a cycle graph and a star graph connected by few edges.

Question 4

Star graph

Let node 2 be the center node. Then

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}, \tilde{A} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}, \tag{7}$$

So

$$\tilde{D} = \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix}, \tilde{D}^{-1/2} = \begin{pmatrix} \frac{1}{\sqrt{2}} & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & \frac{1}{\sqrt{2}} \end{pmatrix}, \tag{8}$$

and therefore we have

$$\hat{A} = \frac{1}{2} \begin{pmatrix} 1 & \frac{1}{\sqrt{2}} & 0 & 0 \\ 1\frac{1}{\sqrt{2}} & 1 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & \frac{1}{\sqrt{2}} & 1 & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & 1 \end{pmatrix} \tag{9}$$

Let us compute first Z_0:

$$
\begin{aligned}
Z_0 &= f(\hat{A}XW_0) \\
Z_0 &= f\left(\frac{1}{2}\begin{pmatrix} 1 & \frac{1}{\sqrt{2}} & 0 & 0 \\ 1\frac{1}{\sqrt{2}} & 1 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & \frac{1}{\sqrt{2}} & 1 & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & 1 \end{pmatrix}\begin{pmatrix}1\\1\\1\\1\end{pmatrix}\begin{pmatrix}0.5 & -0.2\end{pmatrix}\right) \\
Z_0 &= f\left(\begin{pmatrix} \frac{2+\sqrt{2}}{4} \\ \frac{1+3\sqrt{2}}{4} \\ \frac{2+\sqrt{2}}{4} \\ \frac{2+\sqrt{2}}{4} \end{pmatrix}\begin{pmatrix}0.5 & -0.2\end{pmatrix}\right) \\
Z_0 &= \begin{pmatrix} \frac{2+\sqrt{2}}{8} & 0 \\ \frac{1+3\sqrt{2}}{8} & 0 \\ \frac{2+\sqrt{2}}{8} & 0 \\ \frac{2+\sqrt{2}}{8} & 0 \end{pmatrix} \\
Z_1 &= f(\hat{A}Z_0W_1) \\
Z_1 &= f\left(\frac{1}{2}\begin{pmatrix} 1 & \frac{1}{\sqrt{2}} & 0 & 0 \\ \frac{1}{\sqrt{2}} & 1 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & \frac{1}{\sqrt{2}} & 1 & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & 1 \end{pmatrix}\begin{pmatrix} \frac{2+\sqrt{2}}{8} & 0 \\ \frac{1+3\sqrt{2}}{8} & 0 \\ \frac{2+\sqrt{2}}{8} & 0 \\ \frac{2+\sqrt{2}}{8} & 0 \end{pmatrix}\begin{pmatrix}0.3 & -0.4 & 0.8 & 0.5\\ -1.1 & 0.6 & -0.1 & 0.7\end{pmatrix}\right) \\
Z_1 &= f\left(\begin{pmatrix} \frac{10+3\sqrt{2}}{32} & 0 \\ \frac{7+9\sqrt{2}}{16} & 0 \\ \frac{10+3\sqrt{2}}{32} & 0 \\ \frac{10+3\sqrt{2}}{32} & 0 \end{pmatrix}\begin{pmatrix}0.3 & -0.4 & 0.8 & 0.5\\ -1.1 & 0.6 & -0.1 & 0.7\end{pmatrix}\right) \\
Z_1 &= \begin{pmatrix} \frac{30+9\sqrt{2}}{320} & 0 & \frac{10+3\sqrt{2}}{40} & \frac{10+3\sqrt{2}}{64} \\ \frac{21+27\sqrt{2}}{160} & 0 & \frac{7+9\sqrt{2}}{20} & \frac{7+9\sqrt{2}}{32} \\ \frac{30+9\sqrt{2}}{320} & 0 & \frac{10+3\sqrt{2}}{40} & \frac{10+3\sqrt{2}}{64} \\ \frac{30+9\sqrt{2}}{320} & 0 & \frac{10+3\sqrt{2}}{40} & \frac{10+3\sqrt{2}}{64} \end{pmatrix}
\end{aligned}
\tag{10}
$$

This computation shows that the 3 satellite nodes have equal embedding and the center node has a different embedding. This is expected since they have same edge structure.

Cycle graph

Let us now compute the features in the cycle graph case:

$$
\begin{aligned}
Z_0 &= f(\hat{A}XW_0) \\
Z_0 &= f\left(\frac{1}{3}\begin{pmatrix}1&1&0&1\\1&1&1&0\\0&1&1&1\\1&0&1&1\end{pmatrix}\begin{pmatrix}1\\1\\1\\1\end{pmatrix}\begin{pmatrix}0.5 & -0.2\end{pmatrix}\right) \\
Z_0 &= f\left(\begin{pmatrix}1\\1\\1\\1\end{pmatrix}\begin{pmatrix}0.5 & -0.2\end{pmatrix}\right) \\
Z_0 &= \begin{pmatrix}0.5&0\\0.5&0\\0.5&0\\0.5&0\end{pmatrix} \\
Z_1 &= f(\hat{A}Z_0W_1) \\
Z_1 &= f\left(\frac{1}{3}\begin{pmatrix}1&1&0&1\\1&1&1&0\\0&1&1&1\\1&0&1&1\end{pmatrix}\begin{pmatrix}0.5&0\\0.5&0\\0.5&0\\0.5&0\end{pmatrix}\begin{pmatrix}0.3&-0.4&0.8&0.5\\-1.1&0.6&-0.1&0.7\end{pmatrix}\right) \\
Z_1 &= f\left(\frac{1}{1}\begin{pmatrix}1&0\\1&0\\1&0\\1&0\end{pmatrix}\begin{pmatrix}0.3&-0.4&0.8&0.5\\-1.1&0.6&-0.1&0.7\end{pmatrix}\right) \\
Z_1 &= \frac{1}{2}\begin{pmatrix}0.3&0&0.8&0.5\\0.3&0&0.8&0.5\\0.3&0&0.8&0.5\\0.3&0&0.8&0.5\end{pmatrix}
\end{aligned}
\tag{11}
$$

As a conclusion, the four nodes have the same embedding, this is expected since they all play a symmetric role each other, they are indistinguishable because they have same edge structure and same feature initialisation.

Conclusion

It is interesting to remark that the second feature zero for both graphs, and this is true starting from the first layer until the end of the second layer. This is due to applying the Relu on a negative weight of W_0.

Task 13

```
Epoch: 100 loss_train: 0.1348 acc_train: 0.9532 loss_val: 0.7170 acc_val: 0.8413 time: 0.0200s
Optimization Finished!
Total time elapsed: 2.1870s

Test set results: loss= 0.4701 accuracy= 0.8708
```

According to scikit-learn's documentation (`https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html`) "It is highly recommended to use another dimensionality reduction method (e.g. PCA for dense data or TruncatedSVD for sparse data) to reduce the number of dimensions to a reasonable amount (e.g. 50) if the number of features is very high. This will suppress some noise and speed up the computation of pairwise distances between samples."
Since $n_{features} = 32, n_{samples} = 542$, we don't need to use PCA to reduce the dimension before using TSNE, we can see the 2D visualisation in 3.

References

[1] Yves Bouteiller. Skip-gram neural network for graphs. `https://towardsdatascience.com/skip-gram-neural-network-for-graphs-83b8f308bf87`. Accessed: 2023-11-20.

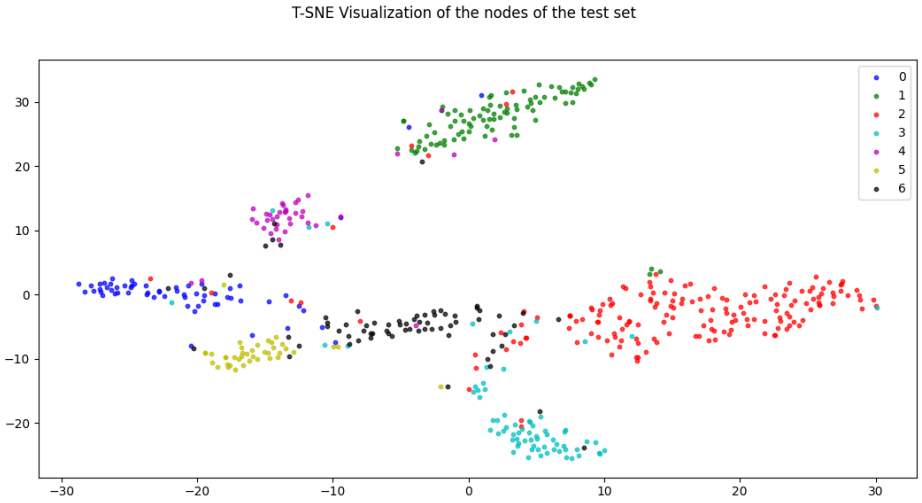

Figure 3: Visualisation of the dimension-32 features learnt by the GNN and extracted from the second message passing layer in a 2-dimensional space using T-SNE.

[2] William L. Hamilton. Graph representation learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 14(3):1–159.

[3] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[4] Donghui Yan, Ling Huang, and Michael I. Jordan. Fast approximate spectral clustering. In *Knowledge Discovery and Data Mining*, 2009.