Balthazar Neveu
balthazarneveu@gmail.com

Lab session # 5
ALTEGRAD 2023

11/20/23

Code

More info: MVA ALTEGRAD Balthazar Neveu on Github

1 Unsupervised node embeddings using Deepwalk

We're trying to construct meaningful embeddings for a graph made of crawled french websites.

- Number of nodes: 33226 . Each node is a website link.
- Number of edges: 354529 . Edges means the presence of hyperlinks between 2 websites.
- Dimension of embeddings of Word2Vec: 128
- Random walk length $L = 20$
- At each node, $n_{walks} = \gamma = 10$ will be pre-stored.

Task 1 & 2: Details on random walks sampling implementations

Unexpectedly, initial naïve implementation took way too much time. Profiling led me to spot two bottlenecks:

- I should not use *np.random.choice* but rather *random.randint* to sample the next node in the walk among neighbors.
- In addition, I used a dictionary to pre-store the neighbors instead of using the remove redundant calls to *list(G.neighbors(node))*
- In this dataset, nodes are not indexes (e.g. integers) but string (hyperlinks). I tried to relabel to ints but it didn't really sped up and addressing the first two points was enough.

```
# Create a mapping from strings to integers
node_mapping = {node: i for i, node in enumerate(G.nodes())}

# Create a reverse mapping if you need to convert back to strings later
reverse_mapping = {i: node for node, i in node_mapping.items()}
G_int = nx.relabel_nodes(G, node_mapping)
```

Task 4 : Node embdeddings visualization

Refer to 1

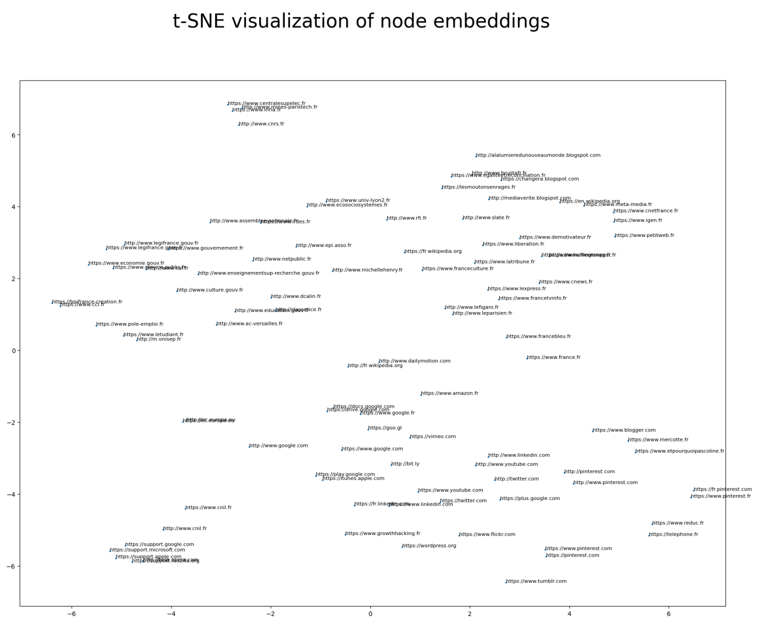

Figure 1: t-SNE is used on the high (128) dimensional node embeddings to reduce dimensionality to 2D coordinates, which allows visualization of a few samples (here $n = 100$) - embdeddings look relevant from the name of the website e.g. Pinterest, Tumblr, Wordpress appear in the same location for instance.

Question 1

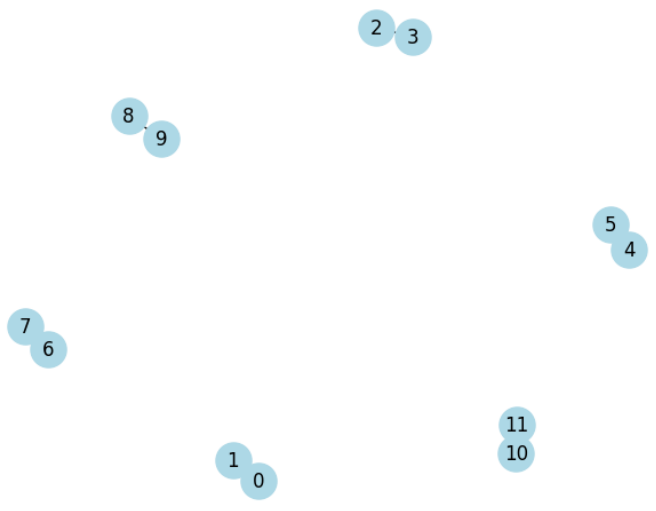

Figure 2: Visualization of the $M = 6$-K_2 components graph.

Nodes within Connected Components

For nodes within each K_2 component, we expect a high cosine similarity in their embeddings.
In a K_2 graph, the only possible walks are between the two connected nodes.
The DeepWalk algorithm, which relies on these walks to generate embeddings, will frequently observe these two nodes in proximity. Consequently, their vector embeddings will be very similar, reflecting their direct and exclusive connection in the graph.

Nodes in Different Connected Components

In contrast, nodes in different K_2 components are expected to exhibit lower cosine similarity in their embeddings.

Each K_2 is an isolated connected component, implying that random walks within one K_2 component do not include nodes from another K_2 component.
As a result, the embeddings generated by DeepWalk will capture this lack of relation in the graph structure, leading to lower similarity in the vector space for nodes from different components.
In 3, I did a quick experiments to validate this result. This reveals the relavance of the features. Notebook is available to reproduce these experiments.

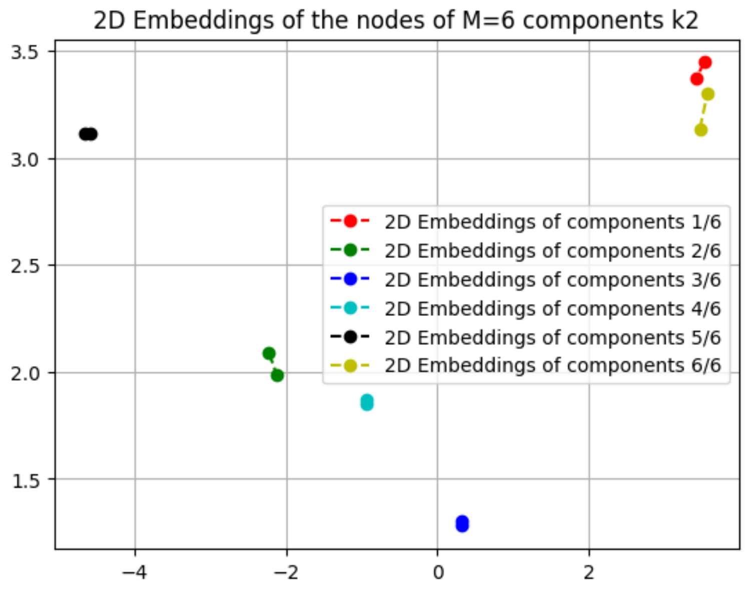

Figure 3: Relevant 2D embeddings from DeepWalk for M-K_2 components. Nodes inside the same components (same color) have very similar 2D embeddings whereas nodes from different components are farther away.

2 Node classification

Task 5: visualization of Zachary's karate club

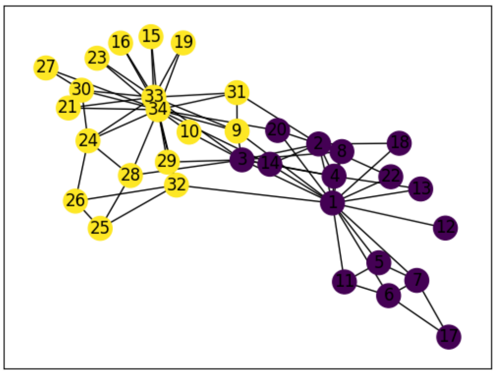

Figure 4: Visualization of the Karate graph with 2 labels (groundtruth)

Task 6/7/8: Classification of Zachary's karate club

```
Deep walk embeddings: accuracy = 1.000
Spectral embeddings accuracy =  0.857
```

These results are given for a training ratio of 80% on a single training/validation set split. A more fair evaluation reveals an accuracy of 99% with Deep walk embeddings accuracy and 79% with Spectral embdeddings, in average over 1000 runs, with a training data ratio of 80%, please refer to 5.

Comparison of DeepWalk embeddings and spectral embeddings

The code for this part can be found in *code/part1/classifiers_comparisons.py*
Unsupervised deepwalk embeddings are much more powerful than spectral embeddings when evaluated on the weakly supervised classification task as shown on 5. *Please keep in mind that as the training ratio increase,*

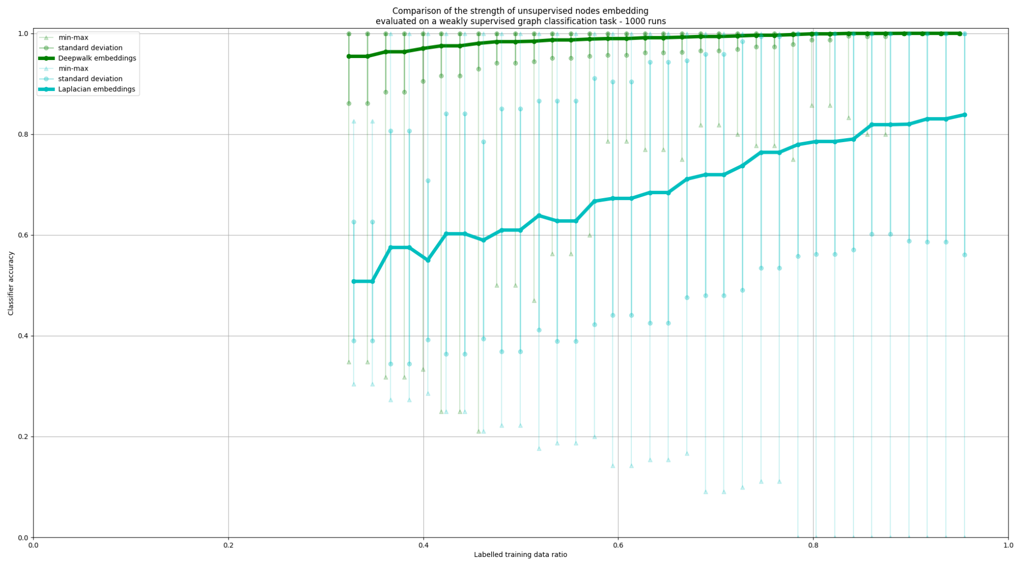

Figure 5: Average classification accuracy over 1000 runs (various seeds) for various training ratios. min & max accuracy and standard deviation over the 1000 runs are reported.

we also get less data for evaluation. Accuracies reported for 95% of training data leave 5% if unlabelled data for evaluation, thus not making a really fair job. It is impressive to see that with only 30% of the labels being given, deep walk embeddings based classifier is able to classify the remaining 70%unlabelled nodes with an accuracy of 95%. Spectral features give 50% accuracy in this same regime, meaning they're nearly not informative in this case.

Question 2

Assume we have a graph $G \in (V, E)$ where $|V|$ denotes the number of nodes (a.k.a vertices).
For complexity study, we're interested on evaluating how the algorithm scales with the size of the graph... and indeed we'll study complexity regarding the number of nodes $|V|$.

- **Skip-gram embeddings**: $O(|V| log_2(|V|))$
- **Spectral embeddings**: $O(|V|^3)$ *pessimistic bound when we cannot make assumptions on the graph sparsity.*

Deep walk

Building the training dataset

- Building the γ walks of length L for each of the $|V|$ nodes costs $O(L.\gamma.|V|)$.
- Note: *Memory footprint* $\propto L.\gamma.|V|$. *when we pre-store all random walks indexes at once... Since SkipGram is going to use stochastic gradient descent, we could sample the batches of random walks on the fly to keep a low memory footprint (it does not seem like a total requirement to sample a frozen dataset of random walks except simplicity of the code)..*
- This creates a training corpus of $k.|V|$ sentences of L words each. Each word corresponds to a node index (vocabulary size = $|V|$) [3]

Skip gram $O(|V|log_2(|V|))$

In the deep walk use of the skip-gram model designed originally for language, here are the specificities:

- Vocabulary size corresponds to the number of nodes $|V|$.
- Central word(node) is predicting a context of $2.w$ words around (order is ignored).
- $2.w + 1 \leq L$ (window is smaller than the length of the walks).

I've made the effort to break down complexity to get something less rough $O(|V|.log(|V|))$ (meaning what's the actual rough factor in front of n.log(n)).

From the Word2Vec paper [2], the skip-gram complexity for each word is $O(2.w.(d + dlog_2(|V|)))$
Let's have a quick description of the skip-gram model and break down its complexity For each window of $2.w + 1$ words, center word v needs to predict $2.w$ context words.

- The input center word v will be mapped to the hidden vector d (simple lookup in the $|V|.d$ matrix). This costs O(d).
- In the non optimized version, the hidden vector has to be mapped $2.w$ times to a $|V|$ dimensional vector (1 logit for each element in the vocabulary).

  - The matrix of weights is shared for all $2.w$ words (words considered unordered).

  - This fully-connected projection matrix is of size $d.|V|$, going from hidden dimension d to logits in dimension $|V|$

  - Then it has to be followed by a Softmax operation over $|V|$ dimensions.

  - This would lead to a huge cost per word v of $2.w(d + d.|V|)$.
- in Word2Vec [2] section 3.2, the authors make use of the hierarchical softmax which allows to lower complexity from $O(2.w.(d + d|V|))$ down to $O(2.w.(d + dlog_2(|V|)))$

  - log_2 comes from using binary trees.

  - Please keep in mind that at inference time, we're only interested in the hidden vector and that this heavy computation is only needed during training.

Assume we have ν epochs (one epoch means processing all $\gamma.|V|$ sentences),
We make a rough assumption that we'll sample a single window of length $2w + 1$ along each walk of length L, *otherwise an extra factor can be considered.*
Skip gram training total complexity is $O(\gamma.|V|.\nu.2.w.(d + d.log_2(|V|)))$

Summing it up, we get a total complexity of

$$O(\gamma.|V|(L + \nu.2.w.(d + d.log_2(|V|)))$$

Most of the computation cost in the Skip-Gram training and the complexity is $\propto 2.w.\gamma.\nu.d.|V|.log_2(|V|)$ which scales in $O(|V|.log_2(|V|))$ with the size of the graph.
Please note that $2.w.\gamma.\nu d.$ are basically users choice/hyper-parameters but they're non negligible factors in practice.

Identifying parameters (orders of magnitude) in the Karate Graph

- $|V| = 34$ nodes : *This very small graph makes training deepwalk in seconds actually.*
- $\gamma = 10$ walks at each node.
- $L = 20$ (length of the walks).
- $d = 128$ is the hidden dimension used in the skip-gram.
- $\nu = 5$ epochs.
- $w = 8$ for the window length used in gensim's Word2Vec implementation.
- $|E| = 78$ edges *sparsity is a nice property to lower down the complexity of spectral embeddings computation*

Remark on sparsity $|E| << |V|^2 = 1156$ (corresponding to a clique, adjacency matrix is definitely sparse). Having sparse adjacency is useful in the context of spectral embeddings to reduce complexity.

Spectral embeddings $O(|V|^3)$

- Regarding time complexity, the primary factor in spectral embedding is the eigendecomposition of the Laplacian matrix. Typically, this process has a time complexity of $O(|V|^3)$ for a dense matrix of size $|V| \times |V|$. where $|V|$ represents the number of nodes in the graph.
- Worst case scenario is a clique (dense).
- But in practice, graphs are sparse and complexity of eigen decomposition can be reduced. In the code for instance, we use the scipy sparse eigen decomposition.

3 Graph Neural Networks

Question 3

We're looking at the scenario where no self-loops were added to the graph (e.g. if we'd work with the raw adjacency matrix after normalization $D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$). This is a bad idea and we'll try to explain why.

- Without adding the self-loop, you're doing vanilla message passing from neighboring node and forget to consider the node's own feature.
- The node's hidden state is solely a function of its neighbors' features.
- The node's own features are not considered directly, potentially losing the unique characteristics of the node itself. This might lead to less effective representations. After a single layer, the feature of a node will simply be replaced by the smoothed version of neighboring features.
- It seems from the Kipf & Welling paper [1] that the self loops also bring numerical stability.

Renormalization trick

In [1], the authors introduced the use of $\hat{A}$ explaining there are good numerical reasons aswell. In the basic formulation, $D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$, it seemed at first sight that most of the normalization had been taken care of (basically do not use the adjacency matrix and aggregate by sum but rather something much similar to averageing over neighbors). But there seems to be more.
In practice, I conducted some checks on the clique (complete) graph.

- For the clique, $D^{-\frac{1}{2}}AD^{-\frac{1}{2}} \propto 1^{|V|,|V|} - I$.

  This matrix has 1 eigen value equal to 1, the rest are $|V| - 1$ negative eigen values. $(-\frac{1}{|V|-1})$

  If we apply several layers of convolution (assume that linear layers are just doing identity), these negative values may lead to numerical instabilities (oscillations basically).
- For the clique, $\hat{A} \propto 1^{|V|,|V|}$ which is of rank 1, having a single non null eigenvalue (equal to 1 here) and all other eigen values are 0. Having this self loop seems like a good idea.

```
# Clique size 8 - Eigen values
Self loops True: [0.00e+00 1.00e+00 1.892e-33 1.892e-33 5.278e-34 1.2655-33]
Self loops False: [ 1.         -0.14285714 -0.14285714 -0.14285714 -0.14285714 -0.14285714]
```

Please refer to *code/part2/self_loops_in_adjacency_matrix_question_3.ipynb* for the code.

Loosing sight of the original feature

Single layer case

Consider a single layer network, removing the self-loop makes the learning task more difficult as the node feature itself is not used, only its neighbor features are used and aggregated.

Take the example of the **star graph**.

- Satellite nodes become indistiguishable as they will only receive the same message from the central node.
- After the first convolution layer, all satellite nodes have the same feature.
- Add the self-loop back and all satellite nodes are different and keep a trace of their input feature.

Intuitively, the node feature may contain unique characteristics: if we provide one-hot encoded vectors (like we do in task 11) as input features to a standard classifier (ignoring the graph aspects e.g. setting the adjacency graph to identity), we could memorize most of the training graph labels at training time. Test accuracy would be low though as the graph structure is ignored. These self loops allow passing the current node feature to the next layer.
Note: *nn.Linear(n_feat, n_classes)* is able to totally overfit and memorize the training node labels when providing input features. Putting a mix of message passing from the neighbors and this memorization ability makes a GNN sucessful.

Two layer case

For the star graph case

- The center node will recover its original feature (as the average over all satellite nodes features which were made identical afterthe first layer).
- After the second convolution layer, all satellites nodes have the same feature (broadcast of the central node feature).

Information seems to go back and forth. This feels like an unstability in this toy example where the locality of the information is not preserved.

Task 11: GNN Classification on karate club - one-hot encoded input features 100% accuracy

```
Epoch: 100 loss_train: 0.0016 acc_train: 1.0000 time: 0.0152s
Test set results: loss= 01 accuracy= 1.0000
```

Task 12: GNN Classification on karate club - identical input features 28.6% accuracy

```
Epoch: 100 loss_train: 0.6655 acc_train: 0.6667 time: 0.0059s
Test set results: loss= 0.7871 accuracy= 0.2857
```

Network cannot make a distinction between the input nodes features.
For instance, it prevents the network from memorizing a mapping between the training node "indexes" and the output class. It inherently forces the network to recognize structure patterns from the edges to label the nodes both at training and inference time... which makes the classification task almost impossible (it could work if the 2 clusters had 2 very different structures, like a star graph and a clique connected to each other with like 1 link, there could be a chance identical features would allow classifying the 2 groups.)
In the case of on-hot encoded input features, at training time, the network can memorize that some labels correspond to a given class and later at inference time, convolution allow to spread this information to unlabelled nodes... wich results in far better results.

Task 13:Supervised GNN Classification on Cora 87.1% accuracy

```
Epoch: 100 loss_train: 0.1436 acc_train: 0.9550 loss_val: 0.4949 acc_val: 0.8745 time: 0.0130s
Test set results: loss= 0.5189 accuracy= 0.8708
```

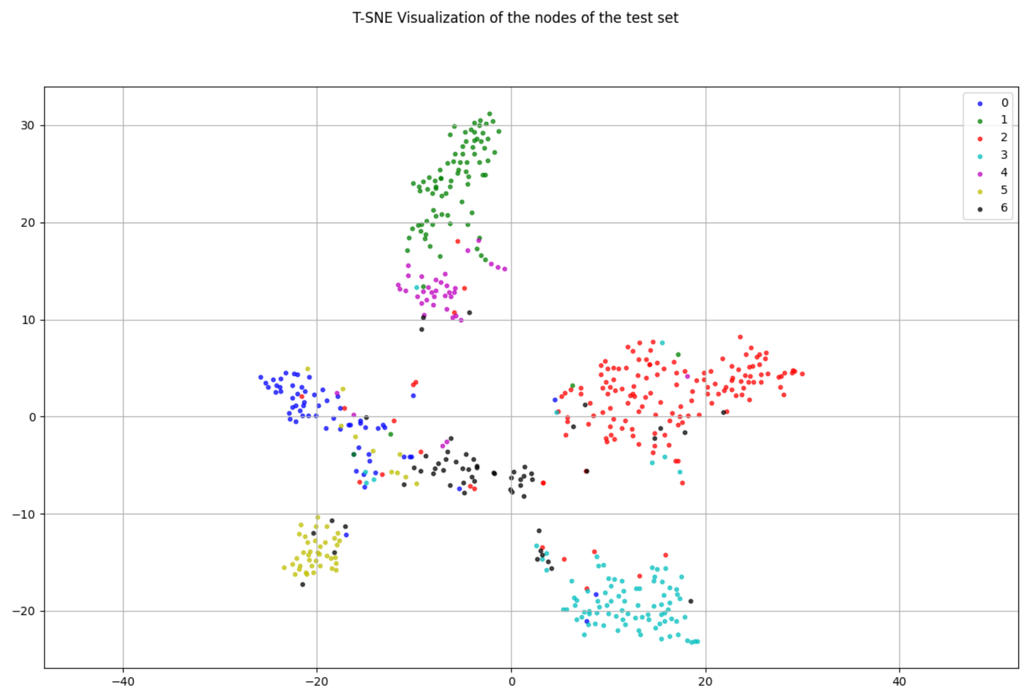

Figure 6: Visualization of the discriminative features learnt by the GNN (second hidden linear layer). High dimensional (32) hidden feature vectors of 542 test nodes are visualized in a low 2D dimension space using T-SNE

Question 4

Methodology

Rational approach is to use Sympy to ease the following computations.
Here's the example for the star graph below.
You can find a short dedicated Notebook *code/part2/symbolic_question_4.ipynb*

```
from sympy import symbols, Matrix, sqrt, Rational, latex, evaluate, N, Max

# Input features
X = Matrix([[1], [1], [1], [1]])

# First layer weights
W_0 = Matrix([[Rational(1, 2), -Rational(2, 10)]])


### STAR GRAPH
# Define the variable u
u = 1/sqrt(2)
# Normalized adjacency matrix (computed semi manually)
A = Rational(1, 2) * Matrix([
    [Rational(1, 2), u, u, u],
    [u, 1, 0, 0],
    [u, 0, 1, 0],
    [u, 0, 0, 1]
])

# First layer hidden unit
Z_0_before_relu = A*X*W_0
Z_0 = Z_0_before_relu.applyfunc(lambda x: Max(0, x))

# Second layer weights
# Note: second row of W_1 will not be used as Z_0 second column is full of zeros
W_1 = Matrix([
    [Rational(3,10), -Rational(2,5), Rational(4,5), Rational(1,2)],
    [-1.1, 0.6, -0.1, 0.7]
])
Z_1 = (A*Z_0*W_1).applyfunc(lambda x: Max(0, x))
```

Here's an additional validation to check that the analytic expressions match with the numerical computation

```
from utils import normalize_adjacency
import networkx as nx
import numpy as np
G = nx.star_graph(3)
Anorm = normalize_adjacency(nx.adjacency_matrix(G))
X = np.ones((4, 1))
W0 = np.array([[0.5, -0.2]])
W1 = np.array([[0.3, -0.4, 0.8, 0.5], [-1.1, 0.6, -0.1, 0.7]])
Z0 = (Anorm@X@W0).clip(0, None)
Z1 = (Anorm@Z0@W1).clip(0, None)
print(f"{Z0} \n{np.array(N(Z_0))}")
print(f"{Z1} \n{np.array(N(Z_1))}")
```

Star graph

Adjacency matrix

$$A_{S^4} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

$$\tilde{A}_{S^4} = A_{S^4} + I = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

$$\tilde{D_{S^4}} = \text{diag}[4,2,2,2] = \begin{bmatrix} 4 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix}$$

$$\tilde{D_{S^4}}^{-\frac{1}{2}} = \text{diag}[\tfrac{1}{2}, \tfrac{1}{\sqrt{2}}, \tfrac{1}{\sqrt{2}}, \tfrac{1}{\sqrt{2}}] = \begin{bmatrix} \frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & \frac{1}{\sqrt{2}} \end{bmatrix}$$

$$\hat{A_{S^4}} = \tilde{D_{S^4}}^{-\frac{1}{2}}.\tilde{A}_{S^4}.\tilde{D_{S^4}}^{-\frac{1}{2}}$$

$$\hat{A_{S^4}} = \begin{bmatrix} \frac{1}{4} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} \\ \frac{1}{2\sqrt{2}} & \frac{1}{2} & 0 & 0 \\ \frac{1}{2\sqrt{2}} & 0 & \frac{1}{2} & 0 \\ \frac{1}{2\sqrt{2}} & 0 & 0 & \frac{1}{2} \end{bmatrix} = \begin{bmatrix} \frac{1}{4} & \frac{\sqrt{2}}{4} & \frac{\sqrt{2}}{4} & \frac{\sqrt{2}}{4} \\ \frac{\sqrt{2}}{4} & \frac{1}{2} & 0 & 0 \\ \frac{\sqrt{2}}{4} & 0 & \frac{1}{2} & 0 \\ \frac{\sqrt{2}}{4} & 0 & 0 & \frac{1}{2} \end{bmatrix} \approx \begin{bmatrix} 0.25 & 0.354 & 0.354 & 0.354 \\ 0.354 & 0.5 & 0 & 0 \\ 0.354 & 0 & 0.5 & 0 \\ 0.354 & 0 & 0 & 0.5 \end{bmatrix}$$

First hidden unit Z^0

Let's use $u := \frac{1}{\sqrt{2}}$

$$\hat{A_{S^4}}.X.W^0 = \frac{1}{4}\begin{bmatrix} \frac{1}{2} + 3u & -? \\ u+1 & -? \\ u+1 & -? \\ u+1 & -? \end{bmatrix}$$

$$Z^0 = \begin{bmatrix} \frac{1}{2} + 3u & 0 \\ u+1 & 0 \\ u+1 & 0 \\ u+1 & 0 \end{bmatrix}$$

I used the $?$ symbol to skip computation because there's no need to perform this computation since these are negative numbers which will be zeroed by $f = \text{ReLU}$

$$Z^0 = \begin{bmatrix} \frac{1}{8} + \frac{3\sqrt{2}}{8} & 0 \\ \frac{\sqrt{2}}{8} + \frac{1}{4} & 0 \\ \frac{\sqrt{2}}{8} + \frac{1}{4} & 0 \\ \frac{\sqrt{2}}{8} + \frac{1}{4} & 0 \end{bmatrix} \approx \begin{bmatrix} 0.655 & 0 \\ 0.427 & 0 \\ 0.427 & 0 \\ 0.427 & 0 \end{bmatrix}$$

Second hidden unit Z^1

To compute Z^1, we have to nottice straight away that the second row of W^1 will be multiplied only by 0

$$W^1 = \begin{bmatrix} \frac{3}{10} & -\frac{2}{5} & \frac{4}{5} & \frac{1}{2} \\ -1.1 & 0.6 & -0.1 & 0.7 \end{bmatrix}$$

Performing computation manually does not make sense here and the room for mistake is huge. We use symbolic calculus to perform the computation correctly.

$$\hat{A_{S^4}}.Z^0.W^1 = \begin{bmatrix} \frac{3}{320} + \frac{9\sqrt{2}}{320} + \frac{9\sqrt{2}\left(\frac{\sqrt{2}}{8}+\frac{1}{4}\right)}{40} & -\frac{3\sqrt{2}\left(\frac{\sqrt{2}}{8}+\frac{1}{4}\right)}{10} - \frac{3\sqrt{2}}{80} - \frac{1}{80} & \frac{1}{40} + \frac{3\sqrt{2}}{40} + \frac{3\sqrt{2}\left(\frac{\sqrt{2}}{8}+\frac{1}{4}\right)}{5} & \frac{1}{64} + \frac{3\sqrt{2}}{64} + \frac{3\sqrt{2}\left(\frac{\sqrt{2}}{8}+\frac{1}{4}\right)}{8} \\ \frac{3\sqrt{2}}{160} + \frac{3}{80} + \frac{3\sqrt{2}\cdot\left(\frac{1}{8}+\frac{3\sqrt{2}}{8}\right)}{40} & -\frac{\sqrt{2}\cdot\left(\frac{1}{8}+\frac{3\sqrt{2}}{8}\right)}{10} - \frac{1}{20} - \frac{\sqrt{2}}{40} & \frac{\sqrt{2}}{20} + \frac{1}{10} + \frac{\sqrt{2}\cdot\left(\frac{1}{8}+\frac{3\sqrt{2}}{8}\right)}{5} & \frac{\sqrt{2}}{32} + \frac{1}{16} + \frac{\sqrt{2}\cdot\left(\frac{1}{8}+\frac{3\sqrt{2}}{8}\right)}{8} \\ \frac{3\sqrt{2}}{160} + \frac{3}{80} + \frac{3\sqrt{2}\cdot\left(\frac{1}{8}+\frac{3\sqrt{2}}{8}\right)}{40} & -\frac{\sqrt{2}\cdot\left(\frac{1}{8}+\frac{3\sqrt{2}}{8}\right)}{10} - \frac{1}{20} - \frac{\sqrt{2}}{40} & \frac{\sqrt{2}}{20} + \frac{1}{10} + \frac{\sqrt{2}\cdot\left(\frac{1}{8}+\frac{3\sqrt{2}}{8}\right)}{5} & \frac{\sqrt{2}}{32} + \frac{1}{16} + \frac{\sqrt{2}\cdot\left(\frac{1}{8}+\frac{3\sqrt{2}}{8}\right)}{8} \\ \frac{3\sqrt{2}}{160} + \frac{3}{80} + \frac{3\sqrt{2}\cdot\left(\frac{1}{8}+\frac{3\sqrt{2}}{8}\right)}{40} & -\frac{\sqrt{2}\cdot\left(\frac{1}{8}+\frac{3\sqrt{2}}{8}\right)}{10} - \frac{1}{20} - \frac{\sqrt{2}}{40} & \frac{\sqrt{2}}{20} + \frac{1}{10} + \frac{\sqrt{2}\cdot\left(\frac{1}{8}+\frac{3\sqrt{2}}{8}\right)}{5} & \frac{\sqrt{2}}{32} + \frac{1}{16} + \frac{\sqrt{2}\cdot\left(\frac{1}{8}+\frac{3\sqrt{2}}{8}\right)}{8} \end{bmatrix}$$

We nottice that the second column is filled with negative numbers.

$$Z^1 \;=\; f(\hat{A_{S^4}}.Z^0.W^1) \;=\; \begin{bmatrix} \frac{3}{320} + \frac{9\sqrt{2}}{320} + \frac{9\sqrt{2}\left(\frac{\sqrt{2}}{8}+\frac{1}{4}\right)}{40} & 0 & \frac{1}{40} + \frac{3\sqrt{2}}{40} + \frac{3\sqrt{2}\left(\frac{\sqrt{2}}{8}+\frac{1}{4}\right)}{5} & \frac{1}{64} + \frac{3\sqrt{2}}{64} + \frac{3\sqrt{2}\left(\frac{\sqrt{2}}{8}+\frac{1}{4}\right)}{8} \\ \frac{3\sqrt{2}}{160} + \frac{3}{80} + \frac{3\sqrt{2}\cdot\left(\frac{1}{8}+\frac{3\sqrt{2}}{8}\right)}{40} & 0 & \frac{\sqrt{2}}{20} + \frac{1}{10} + \frac{\sqrt{2}\cdot\left(\frac{1}{8}+\frac{3\sqrt{2}}{8}\right)}{5} & \frac{\sqrt{2}}{32} + \frac{1}{16} + \frac{\sqrt{2}\cdot\left(\frac{1}{8}+\frac{3\sqrt{2}}{8}\right)}{8} \\ \frac{3\sqrt{2}}{160} + \frac{3}{80} + \frac{3\sqrt{2}\cdot\left(\frac{1}{8}+\frac{3\sqrt{2}}{8}\right)}{40} & 0 & \frac{\sqrt{2}}{20} + \frac{1}{10} + \frac{\sqrt{2}\cdot\left(\frac{1}{8}+\frac{3\sqrt{2}}{8}\right)}{5} & \frac{\sqrt{2}}{32} + \frac{1}{16} + \frac{\sqrt{2}\cdot\left(\frac{1}{8}+\frac{3\sqrt{2}}{8}\right)}{8} \\ \frac{3\sqrt{2}}{160} + \frac{3}{80} + \frac{3\sqrt{2}\cdot\left(\frac{1}{8}+\frac{3\sqrt{2}}{8}\right)}{40} & 0 & \frac{\sqrt{2}}{20} + \frac{1}{10} + \frac{\sqrt{2}\cdot\left(\frac{1}{8}+\frac{3\sqrt{2}}{8}\right)}{5} & \frac{\sqrt{2}}{32} + \frac{1}{16} + \frac{\sqrt{2}\cdot\left(\frac{1}{8}+\frac{3\sqrt{2}}{8}\right)}{8} \end{bmatrix}$$

$$Z^1 \approx \begin{bmatrix} 0.185 & 0 & 0.493 & 0.308 \\ 0.134 & 0 & 0.356 & 0.223 \\ 0.134 & 0 & 0.356 & 0.223 \\ 0.134 & 0 & 0.356 & 0.223 \end{bmatrix}$$

Cycle C^4 graph

Adjacency matrix

$$\tilde{A}_{C^4} = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

This is a circulant matrix. We note that the sum of its rows and columns is always 3

$$\hat{A_{C^4}} = \frac{1}{3} \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

First hidden unit Z^0

$\hat{A_{C^4}}$ has only positive coefficients, W^0 has a negactive coefficient in the second column, we know again that the second column of Z^0 will be full of zeros.
But we nottice that multiplying X by $\hat{A_{C^4}}$ when X is full of ones will result in... X.
$Z^0 = f([W^0_1.X, W^0_2.X])$
Since $W^0_1 > 0$ and $W^0_2 < 0$, we have $Z^0 = [W^0_1.X, 0_4]$

$$Z_0 = \begin{bmatrix} \frac{1}{2} & 0 \\ \frac{1}{2} & 0 \\ \frac{1}{2} & 0 \\ \frac{1}{2} & 0 \end{bmatrix}$$

Second hidden unit Z^1

Using ones and zeros vector notation , $Z_0 = \frac{1}{2}[1_4, 0_4]$
In $X.W^1$, the second row of W^1 will null out.
$X.W^1 = \frac{1}{2}[\frac{3}{10}.1_4, -?, \frac{4}{5}.1_4, \frac{1}{2}..1_4]$
Again, the second column being negative, it will end up negative after the final ReLu.
Notticing again that $\hat{A_{C^4}}.1_4 = 1_4$, we end up with
$\hat{A_{C^4}}.X.W^1 = X.W^1$
$Z_1 = f(\hat{A_{C^4}}.X.W^1) = \frac{1}{2}[\frac{3}{10}.1_4, 0, \frac{4}{5}.1_4, \frac{1}{2}.1_4]$

$$Z_1 = \begin{bmatrix} \frac{3}{20} & 0 & \frac{2}{5} & \frac{1}{4} \\ \frac{3}{20} & 0 & \frac{2}{5} & \frac{1}{4} \\ \frac{3}{20} & 0 & \frac{2}{5} & \frac{1}{4} \\ \frac{3}{20} & 0 & \frac{2}{5} & \frac{1}{4} \end{bmatrix}$$

Structure

- **Cycle**: In $Z^1_{C^4}$, we observe that the features are the same for all nodes. It is indeed intuitive since in a cycle, all the nodes have the same structure and we initialized with ”all ones” identical input features. These won’t be distinguishable.
- **Star**: In $Z^1_{C^4}$, we still nottice that the 3 last rows corresponding to the 3 satellite nodes have the same features... again this is expected as they have the same edge structure (related to the central node).
- **Dead activation**: We note that in both cases, the second feature component (= second column) has been ”zeroed”. And this came from the first layer, where the second component had been zeroed out due to a negative weight in W^0. Activation being zero in the first layer, it propagates later.
- Note that we can’t make generalities here like *one negative coefficient in the weights and you’ll propagate dead neurons* since we do not have biases and also input features are simply identical and set to 1.

References

[1] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017.

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

[3] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '14. ACM, August 2014.