ECSE556 Homework 2 Report

Romain Bazin
McGill University
romain.bazin@mail.mcgill.ca

December 6, 2023

Introduction

In this report, we explore different applications of Machine Learning Algorithms to the analysis of human gene networks, such as the ones present on the HumanNet platform.

Code available here.

1 Graph Theory

Let's prove, through mathematical induction that the mth power of the adjacency matrix A for an unweighted, undirected graph, specifically the (i, j) entry, enumerates the walks of length m from vertex i to vertex j.

Base Case (m=2)

For $m = 2$, the squared matrix A^2 reflects the mth power of the adjacency matrix. The (i, j) element of A^2 is calculated as the dot product of the ith row of A with the jth column of A.

In an unweighted, undirected graph, a '1' in matrix A signifies an edge linking two vertices. Hence, the dot product for the (i, j) entry of A^2 tallies the number of 2-length paths from vertex i to vertex j, with each term in the dot product representing a path via an intermediary vertex. This confirms the base case.

Inductive Step

Suppose that our proposition is valid for $m = k$, which means the (i, j) entry of A^k represents the count of k-length walks from i to j.

Now, we must verify that the proposition is also true for $m = k+1$. Considering $A^{k+1} = A^k \cdot A$, the (i, j) value of A^{k+1} is the result of the dot product of the ith row in A^k and the jth column in A.

Each dot product's term is an interaction between the count of walks of length k from vertex i to an intermediary vertex v, and the matrix entry a_{vj}, which is 1 if an edge exists from v to j and 0 otherwise. Effectively, this counts all $k + 1$-length walks from i to j.

As this is true for any vertices i and j, we have demonstrated that if the statement is accurate for A^k, it must also be accurate for A^{k+1}.

Conclusion

By the principle of mathematical induction, we conclude that our statement is accurate for all $m \geq 2$.

2 Random Walks

Given that the stationary distribution π of a random walk on the graph defined is proportional to its degree and to a constant α, we can π express as:

$$\forall i,\ \pi(i) = \alpha \cdot d(i)$$

Where $d(i)$ is the degree of node i, and α is a normalizing constant.
To ensure that π is a probability distribution, we require that it sums to one across all nodes:

$$\sum_i \pi(i) = \sum_i \alpha \cdot d(i) = 1$$

Since the sum of the degrees of all nodes is twice the number of edges in the graph, we have:

$$\alpha \cdot 2m = 1$$

Solving for α gives us:

$$\alpha = \frac{1}{2m}$$

Substituting α back into our expression for $\pi(i)$, we obtain the stationary distribution:

$$\forall i,\ \pi(i) = \frac{d(i)}{2m} \tag{1}$$

To prove that π is stationary, it must verify $\pi = \pi P$.

$$\pi P \iff \forall j,\ \sum_i \pi(i) P_{ij} = \sum_i \frac{d(i)}{2m} \cdot \begin{cases} \frac{1}{d(i)} & \text{if } ij \in E \\ 0 & \text{otherwise} \end{cases}$$

As only the neighbors of j contribute to the sum, and each neighbor i contributes $\frac{1}{2m}$, we have $d(j)$ such terms, thus:

$$\forall j,\ \sum_i \pi(i) P_{ij} = d(j) \cdot \frac{1}{2m} = \frac{d(j)}{2m} = \pi(j)$$

Hence, $\pi = \pi P$ which confirms that π is the stationary distribution.

We have demonstrated that for the simple random walk as defined, the stationary distribution for each node is directly proportional to its degree, as articulated in Equation 1.

3 Random Walk Without Restart

In this section, we have empirically tested the effectiveness of the *Random Walk Without Restart* (RWWR) method in generating node embeddings for graph neural networks. We began by constructing an adjacency matrix from the "HumanNet Co-Expression of Human Genes (hn_HS_CX)" dataset, available here. Edge weights were assigned based on the co-expression values between two genes, using the average value for edges that appeared multiple times. We also removed self-loops and small disconnected components (those with fewer than 4 nodes) from the graph. This resulted in a clean, weighted, undirected graph with 10825 nodes.

Next, we initiated three separate RWWR processes, each starting from a different random node. We allowed these processes to run until the node visitation probabilities stabilized, resulting in three distinct stationary distributions. To analyze these distributions, we created a heatmap, as shown in Figure 1. This heatmap stacks the three distributions for comparison: the horizontal axis represents the nodes, the vertical axis shows the three distributions, and the color intensity indicates the visitation probability of each node during the RWWR.

Our analysis revealed a striking similarity among the three stationary distributions, indicating that RWWR struggles to capture the local information of the starting node. This leads to a homogenization of node embeddings, which undermines their usefulness in learning graph representations, as the embeddings become virtually identical regardless of the starting node.

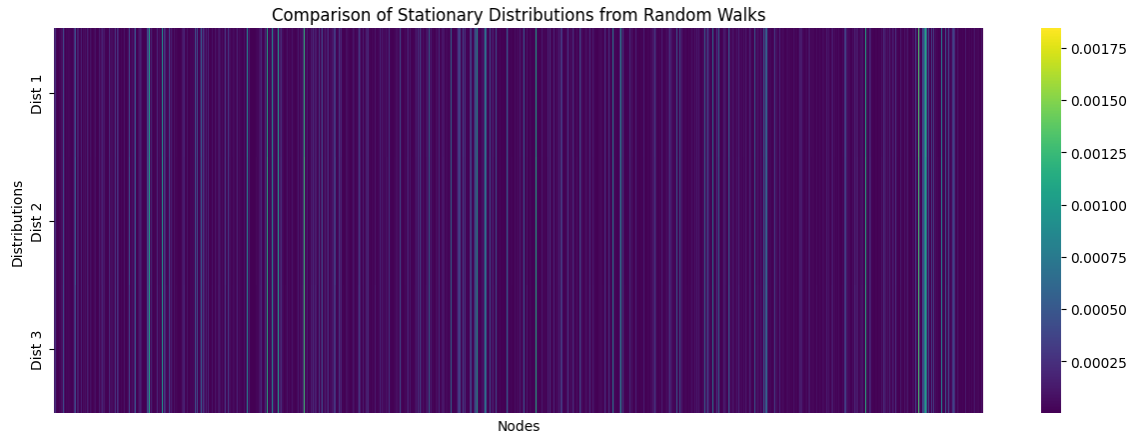


Figure 1: Comparison of the stationary distributions resulting from three RWWR runs with different starting nodes. All runs end up with the same stationary distribution.

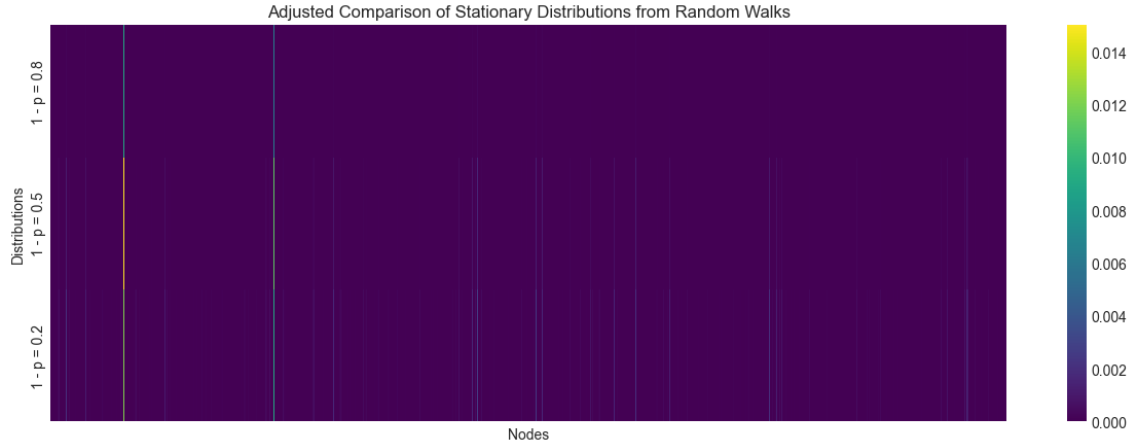


Figure 2: Stationary distributions of three different experiments with fixed randomly selected starting node and fixed restarting node N_1. Node N_1 was removed from the visualization.

4 Random Walk With Restart

In the context of the Random Walk with Restart (RWR) framework, we examine the influence of restart probability and the choice of the restart node on the stationary distribution of a random walk on a network. This network is represented by the transition matrix P, and the RWR process is defined by the iterative equation:

$$q_{k+1} = p \cdot q_k P + (1 - p) \cdot v \tag{2}$$

where q_k is the probability distribution of the walk at step k, p is the probability of following the transition matrix P, and v is a binary vector representing the restart set. For our experiment, v is a binary vector with a single non-zero entry corresponding to the chosen restart node.

As suggested by the assignment, we fixed node N_1 and N_2 to the two nodes of the first row of the dataset. Because these nodes belong to the same row of the dataset, they are connected. The RWR was first conducted with N_1 as the restart node and with different restart probabilities p of 0.2, 0.5, and 0.8, each converging to a stationary distribution from a fixed random initial distribution q_0. Results can be observed on figure 2.

The heatmap illustrates the stationary distributions for the different restart probabilities. Despite the variance in p, a pattern emerges showing that the stationary distributions are not uniform but instead are proportional to each other. The proportionality is more pronounced as p increases, suggesting that the random walk explores the network more broadly, thus reducing the influence of the restart node over the distribution.

The cumulative distribution functions (CDFs) for different restart probabilities, as visualized in figure 3, provide a quantitative perspective on how a random walk's restart probability affects its

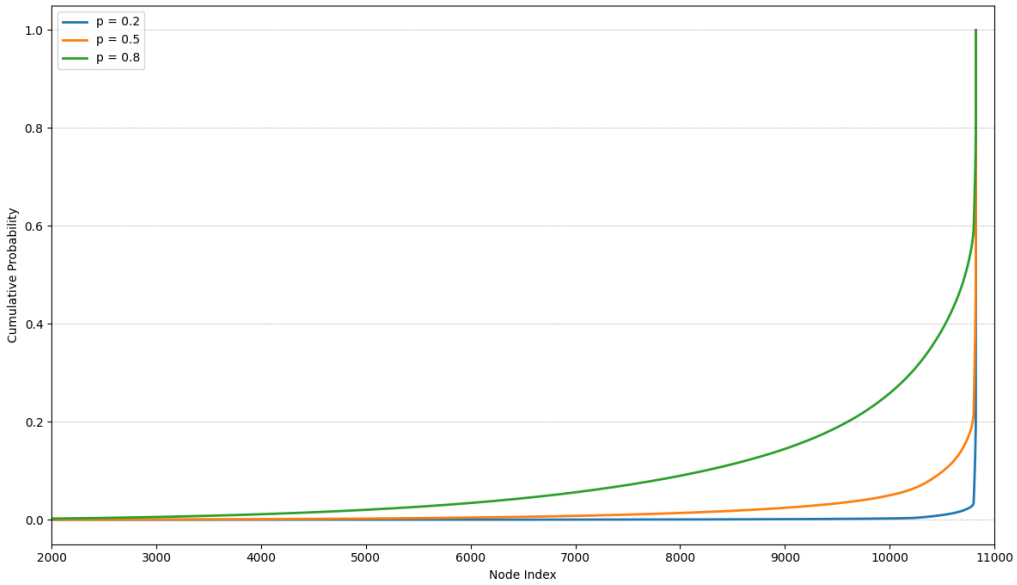


Figure 3: Cumulative Distribution Function (CDF) of RWRs with different values of p. Restarting node was fixed to N_1 and starting node to 1022 (randomly selected).

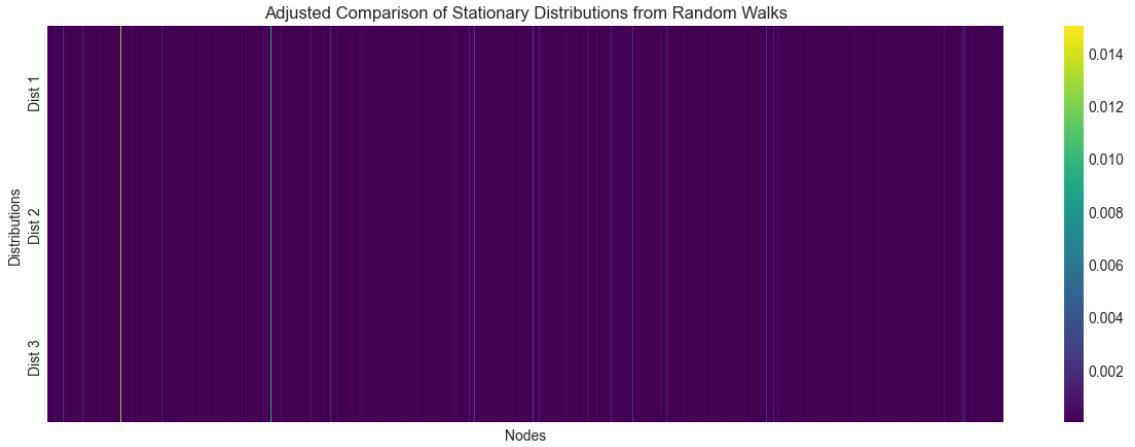


Figure 4: Stationary distributions of three experiments with different randomly selected starting nodes, fixed restarting node N_1, and continuation probability $p = 0.2$. Node N_1 was removed from the visualization. All runs end up with the same stationary distribution.

stationary distribution across the network. The CDF corresponding to $p = 0.2$ remains relatively flat for a large range of node indices, indicating that the walk is more likely to be confined to the vicinity of the restart node, thus reinforcing the local neighborhood structure within the network. As p increases to 0.5 and 0.8, the CDFs begin to ascend more steeply, reflecting a more dispersed and uniform distribution. This suggests that higher values of p correspond to a random walk that covers a wider expanse of the network before restarting, thereby diminishing the restart node's centrality in determining the walk's trajectory.

The steepness of the CDF curve in the tail region for $p = 0.8$ particularly denotes that the walk reaches a larger set of nodes more frequently, which implies a lower degree of restarts and a more global sampling of the network's structure. Consequently, the influence of the restart node diminishes with increasing p, as the walk has a greater chance to venture further into the network. This observation is consistent with the initial hypothesis and underscores the impact of the restart probability on the exploration-exploitation balance in random walks.

To further explore the robustness of the Random Walk with Restart (RWR) model, we modified the initial state of the walk while maintaining the restart node N_1 and the probability p constant. Figure 4 presents the heatmap for the stationary distributions resulting from this variation. Remarkably, the distributions are identical across different starting nodes, which confirms the model's insensitivity to the initial state.

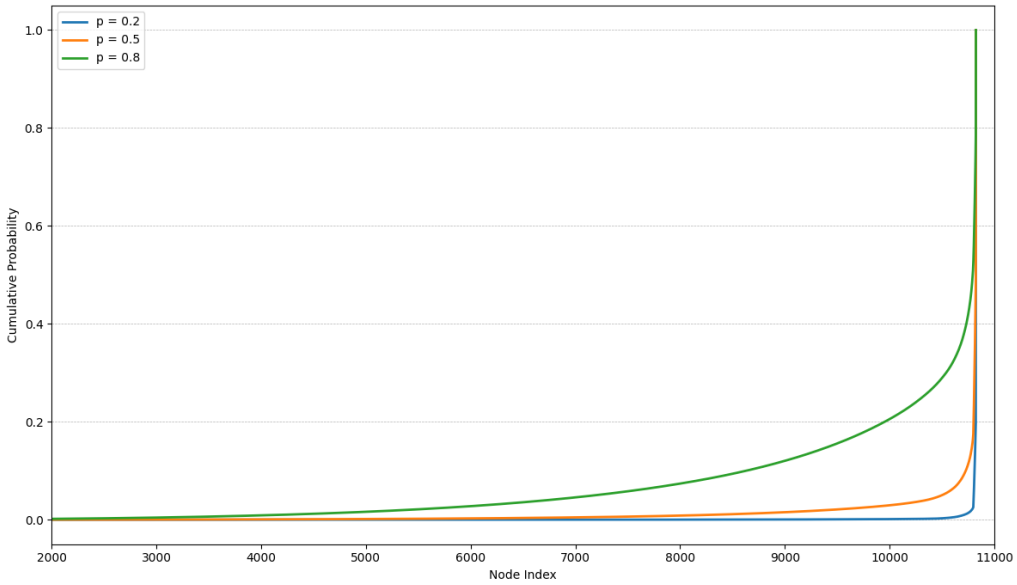


Figure 5: Cumulative Distribution Function (CDF) of RWRs with different values of p. Restarting node was fixed to N_2 and starting node to 1022 (randomly selected).

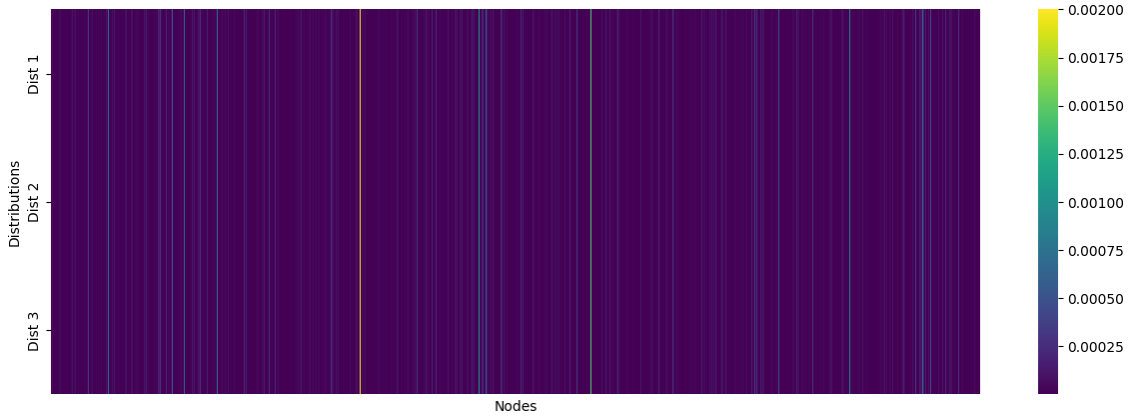


Figure 6: Stationary distributions of three experiments with different randomly selected starting nodes, fixed restarting node N_2, and continuation probability $p = 0.2$. Node N_2 was removed from the visualization. All runs end up with the same stationary distribution.

The experiments with node N_2 as the restart point corroborate our earlier observations, as depicted in figures 5 and 6, demonstrating a consistency in the Random Walk with Restart (RWR) model's performance that is independent of the chosen restart node.

The cumulative distribution functions (CDFs) presented in figure 7 delineate the impact of altering the restarting node on the stationary distributions of a random walk with restart, while keeping the starting node and the continuation probability p constant. The distinct curves for restarting nodes N_1 and N_2 suggest that each restarting node yields a unique stationary distribution, despite identical continuation probabilities and starting points. This diversity in outcomes indicates that the choice of the restart node introduces variability into the walk, preventing the convergence of paths to a singular embedding. This characteristic addresses the concern mentioned in section 3, where a random walk without restart might lead to homogenous embeddings that fail to capture the multiplicity of the network's structure. In conclusion, the random walk with restart enhances the model's ability to generate diverse embeddings, representing a significant advantage over the non-restart approach by allowing for the differentiation of node characteristics based on their role in the network's connectivity.

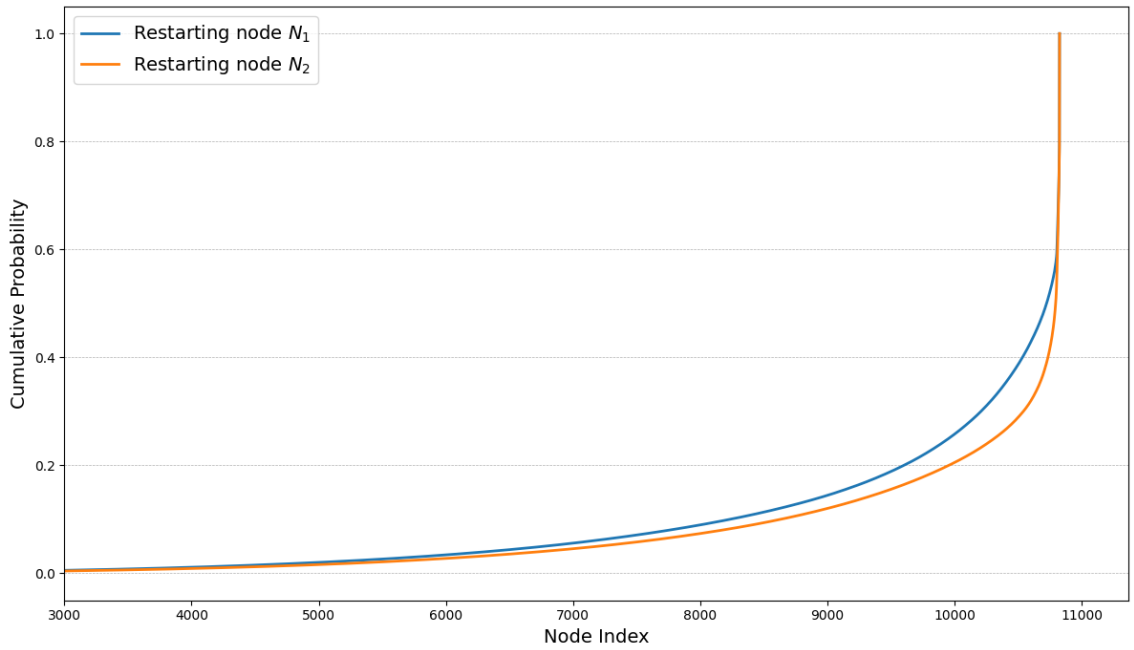


Figure 7: CDFs with fixed probability and starting node, for restarting nodes N_1 and N_2.

5 Node2Vec

In light of the original node2vec implementation being based on an earlier version of Python that is no longer supported, we turned to a more recent Python 3 adaptation available on github. This updated version seamlessly substituted the original, offering the same customizable parameters.

5.1 DFS vs BFS Node Embeddings Similarities

For our experiments, we maintained the standard parameters of the algorithm with the exception of the hyperparameters p and q, which are instrumental in steering the random walks towards a breadth-first or depth-first search pattern:

- **Depth-First Search (DFS) Configuration:** Setting both p and q to lower values, specifically $p = 0.1$ and $q = 0.1$, prompts the algorithm to prioritize exploring further away from the starting node, thus adopting a DFS approach.
- **Breadth-First Search (BFS) Configuration:** In contrast, higher values of p and q, both set to 1 in our case, encourage the algorithm to explore the vicinity of the starting node more thoroughly, aligning with a BFS strategy.

We applied these configurations to produce embeddings for the initial pair of nodes from the "HumanNet Co-Expression of Human Genes (hn_HS_CX)" dataset.

To assess the embeddings' fidelity, we compared them using Pearson and Spearman correlation coefficients, which are indicative of the embedding quality. The correlations, detailed in Tables 8a and 8b, reflect the nuanced effects of our hyperparameter selection on the embeddings.

Correlation	N1 bfs	N1 dfs	N2 bfs	N2 dfs
N1 bfs	1			
N1 dfs	0.051	1		
N2 bfs	0.080	0.138	1	
N2 dfs	0.051	0.128	0.088	1

(a) Pearson correlation coefficients

Correlation	N1 bfs	N1 dfs	N2 bfs	N2 dfs
N1 bfs	1			
N1 dfs	0.046	1		
N2 bfs	0.106	0.161	1	
N2 dfs	0.037	0.119	0.078	1

(b) Spearman correlation coefficients

Figure 8: Correlation coefficients between node embeddings using different configurations

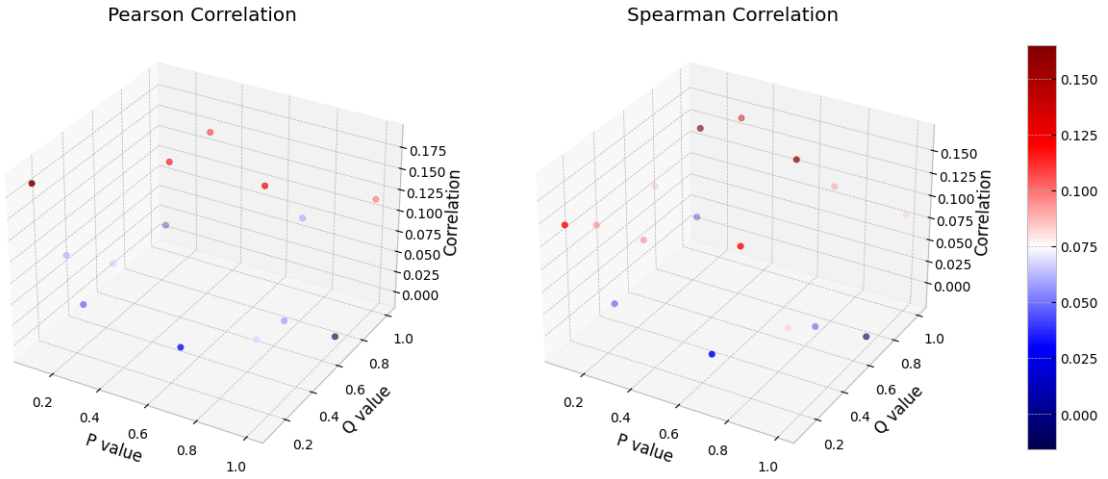


Figure 9: 3D Scatter plot showing the Pearson and Spearman correlations between nodes N_1 and N_2 according to parameters p and q in Node2Vec.

The correlation coefficients presented in Tables 8a and 8b reveal key insights about the node embeddings generated under DFS and BFS configurations in Node2Vec. Both Pearson and Spearman correlations consistently show that embeddings from the same search method (either BFS or DFS) have higher similarity. This underlines the significant impact of the search strategy on node representations in the graph.

Notably, embeddings generated by different configurations (BFS vs. DFS) exhibit lower correlation scores. This divergence highlights how the method of graph traversal (depth-first or breadth-first) leads to distinct node embeddings. For a fixed node, the difference in embeddings by BFS and DFS is pronounced, reflecting the unique characteristics of each traversal strategy.

An intriguing finding is the relatively high similarity score between $N1$ dfs and $N2$ bfs configurations. This might indicate an underlying structural pattern in the "HumanNet Co-Expression of Human Genes (hn_HS_CX)" dataset, captured similarly by both DFS and BFS methods. However, without a deeper investigation into the graph structure, this remains a speculative observation.

5.2 Continuous Nodes Embeddings Similarities according to p and q

The 3D scatter plots in Figure 9 provide a visual exploration of the relationship between the Node2Vec parameters p and q, and the resulting embedding correlations. For both Pearson and Spearman correlations, we observe that as the p value and q value increase, there is a tendency for the correlation to decline. This is indicative of the fact that higher p and q values, which correspond to a BFS-like exploration, yield embeddings that are less correlated with those produced by lower p and q values, akin to a DFS-like exploration.

The color gradient in the scatter plots represents the magnitude of correlation, transitioning from blue (low correlation) to red (high correlation). It is evident that points with low p and q values are colored more intensely, suggesting stronger correlations, particularly for embeddings derived from a similar DFS strategy. In contrast, the embeddings resulting from a BFS approach are scattered across the plot with generally cooler hues, thus lower correlations.

Interestingly, the spatial distribution of points in both plots is not uniform, indicating that the correlation is not solely dependent on the p and q values but is also likely influenced by the inherent structure of the underlying biological network. This graphical analysis corroborates the quantitative findings presented in Tables 8a and 8b.

Note that a limitation of our study is the stochastic nature of Node2Vec's random walks, which introduces variability in the embeddings. Ideally, multiple runs with different seed initializations would provide a more robust analysis, but computational constraints limited this approach. Consequently, the results should be interpreted with caution, acknowledging the potential variability due to the single-run nature of our experiment.

In summary, our findings highlight the importance of the search pattern in Node2Vec for generating node embeddings. The choice of DFS or BFS significantly influences the embeddings, and future studies could benefit from multiple runs to account for the stochastic effects and deepen the understanding of how search strategies impact Node2Vec embeddings.

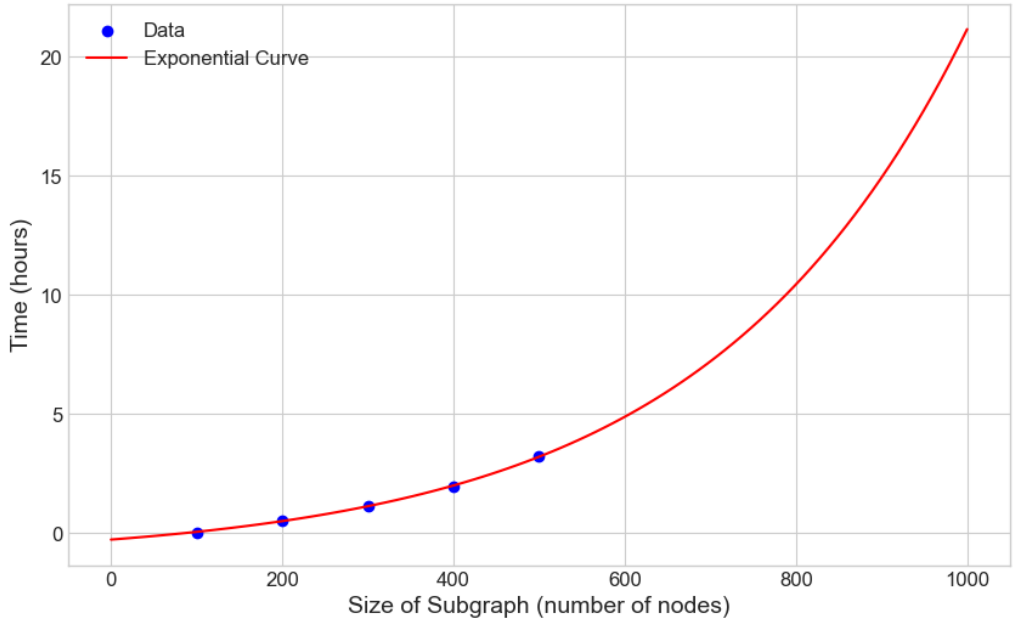


Figure 10: Time to compute the Girvan-Newman algorithm on subgraphs of different sizes. Blue dots represent experimental data.

6 Community Detection

In this analysis, we explore the performance of several clustering algorithms in organizing nodes of a large graph into distinct communities. The algorithms under examination are:

- The Louvain algorithm
- The Clauset-Newman-Moore greedy modularity maximization algorithm
- The Girvan-Newman algorithm

These algorithms were applied in their standard forms from the "networkx" Python library. The graph underwent preprocessing to eliminate self-loops and solitary nodes, which produced a network comprising 10,825 nodes with an average degree of 28.5—an extensive scale for this type of analysis. Due to the considerable magnitude and complexity of the network, attempts at comprehensive visualization yielded results that were not decipherable, thus were not included.

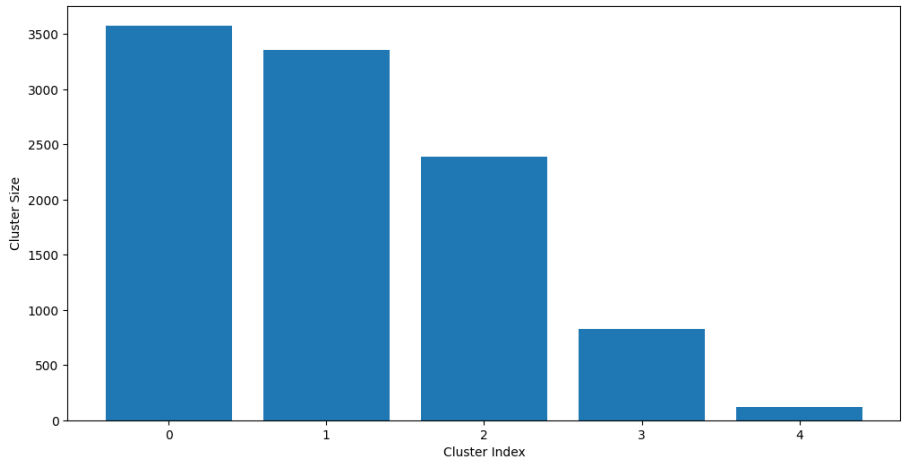
6.1 Girvan-Newman Algorithm

The Girvan-Newman Algorithm posed significant computational challenges due to its iterative edge-removal approach, aimed at disconnecting the most interlinked communities. This process proved to be time-intensive, as illustrated in Figure 10, which depicts the correlation between the algorithm's processing time and the network size. The algorithm's execution involved generating subgraphs centered on a consistently chosen, randomly selected node, expanding in a breadth-first manner. An exponential fit to the observed data allows for extrapolation of the time required for larger network sizes. Notably, a subgraph containing 1000 nodes—merely 10% of the full network—demanded approximately 20 hours of computation on a high-end server CPU. Consequently, applying this algorithm to the entire graph was impractical, and its exclusion from our comparative study was necessary.

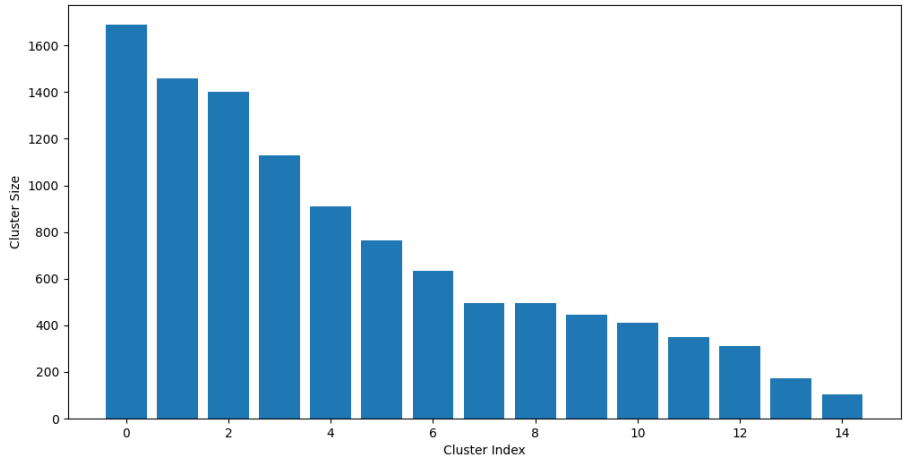
A potential alternative to facilitate the algorithm's inclusion in our analysis could have involved limiting the network to a subgraph of no more than 500 nodes, effectively reducing the original network by 95%. While this reduction would have permitted a comparative evaluation with other algorithms, the resultant subgraph would likely not be representative of the full network's characteristics. Additionally, the outcomes might be overly sensitive to various hyperparameters, such as the choice of the central node and the selection between depth-first or breadth-first expansion strategies when building the subgraph. Given these limitations, we determined that the Girvan-Newman Algorithm was unsuitable for our study and subsequently excluded it from further comparison.

6.2 Louvain vs Clauset-Newman-Moore Comparison

The clustering results were varied. The Clauset-Newman-Moore algorithm produced 120 clusters, many of which were quite small, comprising fewer than 10 nodes. Similarly, the Louvain algorithm identified 22 clusters, with a portion of them also being minimally populated. To ensure a meaningful comparison between the two algorithms, we chose to exclude clusters containing fewer than 100 nodes. This filtration led to the cluster size distribution showcased in figure 11, which provides a more focused view of the larger and potentially more significant clusters. Another filtering strategy would have been to perform clusters fusion, but we didn't delve into these techniques for this simple evaluation. Note that the total number of nodes retained after the applying the filter was above 90%.



(a) Clusters sizes for the Clauset-Newman-Moore algorithm.



(b) Clusters sizes for the Louvain algorithm.

Figure 11: Cluster sizes for different algorithms.

The cluster size distributions obtained from the Louvain and Clauset-Newman-Moore algorithms (figure 11) provide contrasting perspectives on the community structures within the graph. The Louvain method's results, shown in the histogram, indicate a concentration of nodes within a small number of clusters, suggesting the existence of dominant communities. This is indicative of the algorithm's ability to highlight larger, possibly central communities in the network.

On the other hand, the Clauset-Newman-Moore algorithm appears to produce a more balanced partitioning, with a greater number of clusters of moderate size. This could be interpreted as a finer subdivision of the graph, offering a more granular view of its community structure.

To complement these findings, we used the *Jaccard Similarity Score* to quantify the overlap between the communities detected by the two algorithms. The heatmap (figure 12) shows varying degrees of similarity, with some cluster pairs displaying moderate overlap and others very little. This suggests that while there is some agreement between the algorithms on certain community structures, they each have unique interpretations of the network's division into communities.

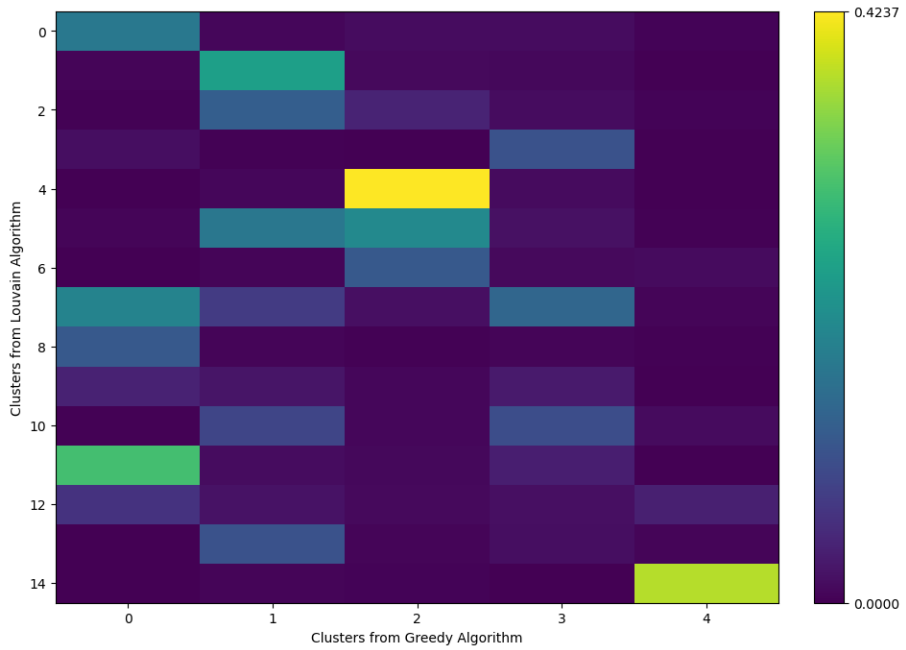


Figure 12: Jaccard similarity between trimmed Greedy and Louvain clusters. Brighter color signifies higher similarity between clusters.

Building upon the distinct cluster size distributions and the Jaccard similarity heatmap analysis, we further evaluated the clustering results using the Rand Score and the Adjusted Rand Score. These scores serve as statistical measures to quantify the agreement between two clustering solutions. Note that we computed both scores on the untrimmed clusters generated by the two algorithms.

The *Rand Score* (RS) is a measure of the similarity between two data clusterings. A score close to 1 indicates a high degree of agreement between two clustering solutions, while a score close to 0 suggests a low degree of agreement. With a Rand Score of 0.748, we observe a moderate agreement between the Louvain and Clauset-Newman-Moore algorithms. This indicates that while the clusterings are not identical, they share a significant number of decisions regarding node groupings.

The *Adjusted Rand Score* (ARS), on the other hand, is a corrected-for-chance version of the Rand Score that adjusts for the possibility of the agreement occurring by chance. The ARS provides a more rigorous assessment of the clustering similarity, with a score of 1 indicating perfect agreement and 0 indicating random clustering. An ARS score of 0.180 suggests that the agreement between the two algorithms is better than random chance but still particularly low. This low ARS score aligns with our earlier observations from the Jaccard heatmap, where we noted only a moderate overlap between the communities identified by the two algorithms.

The interpretation of these scores reinforces our previous conclusions: while there is some consensus between the algorithms on the broader community structures within the graph, they each have their own distinct approach to partitioning the network. The Louvain algorithm tends to group nodes into a smaller number of larger communities, potentially highlighting major hubs within the graph. In contrast, the Clauset-Newman-Moore algorithm distributes nodes across a larger number of smaller clusters, which may be useful for detecting more nuanced community structures.

Given the variation in results, it is clear that no single algorithm provides a definitive solution to community detection. The choice of algorithm should be based on the specific objectives of the analysis and the nature of the graph. For a comprehensive understanding of the community structures in complex networks, an integrated approach that considers the outputs of multiple clustering algorithms could be beneficial. Lastly, the Girvan-Newman algorithm was too computationally expensive to be used in the current scenario of a fairly big network.

7 Exploding / Vanishing Gradients (theory)

7.1 Question 1

The Sigmoid activation function is defined as $\sigma(x) = \frac{1}{1+e^{-x}}$. The derivative, which represents the gradient of the function with respect to its input, is given by:

$$\begin{aligned}
\sigma'(x) &= \frac{d}{dx}\left(\frac{1}{1+e^{-x}}\right) && (3)\\
&= \frac{e^{-x}}{(1+e^{-x})^2} && (4)\\
&= \frac{1}{1+e^{-x}}\left(\frac{e^{-x}}{1+e^{-x}}\right) && (5)\\
&= \frac{1}{1+e^{-x}}\left(1-\frac{1}{1+e^{-x}}\right) && (6)\\
&= \sigma(x)(1-\sigma(x)). && (7)
\end{aligned}$$

In a neural network with multiple layers, the activation a_k of the k-th layer is given by applying the activation function to the weighted sum of inputs, i.e., $a_k = \sigma(\sum_j w_{kj}x_j)$. The gradient of the network function f with respect to the weights and inputs can be computed using the chain rule.

For a network with n layers, let f be the output of the network and a_k be the activation of the k-th layer. The partial the derivative of f with respect to the input x_j is:

$$\begin{aligned}
\frac{\partial f}{\partial x_j} &= \sum_k \frac{\partial f}{\partial a_k}\frac{\partial a_k}{\partial x_j}\\
&= \sum_k \frac{\partial f}{\partial a_k}\sigma'(\sum_j w_{kj}x_j)w_{kj}.
\end{aligned}$$

Under the hypothesis that each layer in the neural network has only one activation value (denoted a_k), one weight value (denoted w_k), and the input to each layer is a single value (denoted x_k), the expressions for the derivatives can be simplified.

In this simplified network model of depth n, the activation a_k of the k-th layer is $a_k = \sigma(w_k x_k)$, where x_k is the output of the previous layer (or the input to the network for the first layer).

The partial derivative of the network function f with respect to the input x is:

$$\begin{aligned}
\frac{\partial f}{\partial x} &= \frac{\partial a_n}{\partial x} && (8)\\
&= \sigma'(w_n x_n)w_n\frac{\partial x_n}{\partial x} && (9)\\
&= \sigma'(w_n x_n)w_n\frac{\partial a_{n-1}}{\partial x} && (10)
\end{aligned}$$

By derivating the sigmoid a second time, using equation 7 we find the maximum value attained by σ' is 0.25 for $x = \frac{1}{2}$.

Given that $\forall\ k, 0 \leq |w_k| \leq 1$,

$$\forall\ k, 0 \leq |\sigma'(w_k x_k)w_k| \leq 0.25 \tag{11}$$

With this last result 11, as well as the chain rule expression depicted in 10, the gradient bounds for the network are:

$$0 < \left|\frac{\delta f(x)}{\delta x}\right| \leq 0.25^n \tag{12}$$

7.2 Question 2

The Leaky Rectified Linear Unit (LeakyReLU) activation function is piecewise linear and is defined as follows:

$$LeakyReLU(x,a) = \begin{cases} x & \text{if } x > 0 \\ ax & \text{if } x \leq 0 \end{cases}$$

where a is a small positive slope to allow gradients to flow even when the unit is not active, typically $0 < a < 1$. The derivative of LeakyReLU is constant and does not depend on x, it is given by:

$$LeakyReLU'(x, a) = \begin{cases} 1 & \text{if } x > 0 \\ a & \text{if } x \leq 0 \end{cases}$$

Therefore, the gradient for any layer in the network is lower bounded by a and upper bounded by 1, which compounds through all layers due to the chain rule 10:

$$a^n \leq \left| \frac{\delta f(x)}{\delta x} \right| \leq 1 \tag{13}$$

Note that the lower bound of the gradients tends to 0 when the depth grows large.

7.3 Question 3

The Mish activation function can be mathematically defined as $f(x) = x \tanh(\ln(1 + e^x))$. The derivative of the Mish function is essential for understanding its behavior in neural networks and is given by:

$$f'(x) = \frac{e^x \omega}{\delta^2}$$

where

$$\omega = 4(x + 1) + 4e^{2x} + e^{3x} + e^x(4x + 6),$$
$$\delta = 2e^x + e^{2x} + 2.$$

Our computational analysis reveals the approximate bounds of the Mish function's derivative to be $[-0.11, 1.09]$. The maximum derivative slightly above 1 suggests a potential for gradient amplification, albeit not necessarily leading to gradient explosion.

Consequently, the gradient's bounds can be expressed by the following equation :

$$0.11^n \leq \left| \frac{\delta f(x)}{\delta x} \right| \leq 1.09^n \tag{14}$$

Much like the lower bound of the gradient for LeakyReLU, which approaches zero as the depth of the network increases (refer to equation 13), we observe a similar trend with our current model. However, in stark contrast, the upper bound of the gradient is not similarly restrained. In fact, it tends to increase with the network's depth, presenting a non-trivial risk of gradient explosion. This characteristic is less than ideal, as it introduces the possibility of instability during the training process.

7.4 Question 4

The analysis of the activation functions above reveals distinct behaviors:

- **Sigmoid Function:** The upper bound of the sigmoid function's gradient tends to 0 as the network depth (n) increases, indicating its susceptibility to the gradient vanishing problem.
- **LeakyReLU:** Despite the derivative of LeakyReLU being upper bounded by 1, it could still potentially be prone to the vanishing gradient problem.
- **Mish Function:** Similar to LeakyReLU, the Mish function's derivative exceeding 1 suggests a degree of resilience against vanishing gradients. However, it also raises concerns about gradient amplification in deep networks, given its maximum derivative value strictly superior to 1.

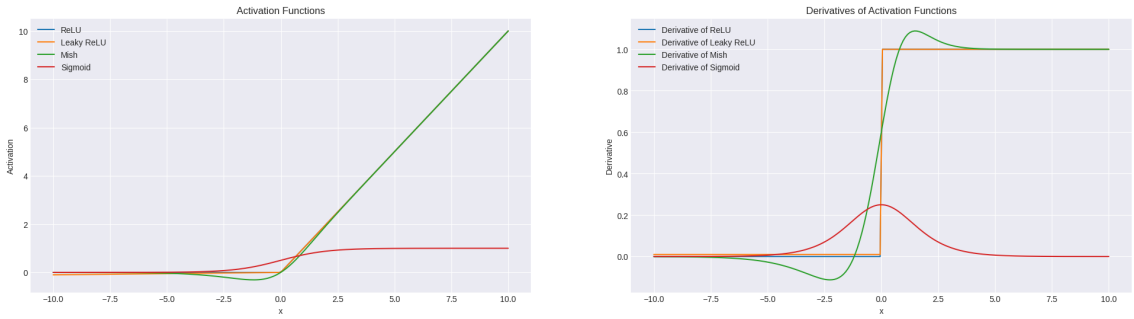


Figure 13: Activation functions and their derivatives.

7.5 Question 5

Regarding the sensitivity to gradient explosion:

- **Sigmoid Function:** Its upper bound, approaching 0 with increasing network depth, renders it less susceptible to gradient explosion, at the cost of increasing the risk for gradient vanishing.
- **LeakyReLU:** Being upper bounded by 1, the activation function seems to have a natural resilience to gradient explosion. However, this function is not derivable on $x = 0$, potentially creating other types of optimization problems related to its non-smoothness, as observed on figure 13.
- **Mish Function:** Like the LeakyReLU funciton, the Mish function posseses a natural resilience to gradient vanishing as its upper bound is above 1. However, not being strictly upper bounded also leaves room for potential gradient explosions. Yet Mish function's smoothness contributes to smoother loss surfaces, potentially leading to more stable and efficient training, as well as robust model performance, as reported by Diganta Misra in *Mish: A Self Regularized Non-Monotonic Activation Function* arXiv:1908.08681, 2020.

In summary, all functions have strength and weaknesses. However, the LeakyReLU and the Mish functions seem to be the best options for training deep networks. Even though the Mish function exhibits beneficial properties like smoothness and a consistent non-zero gradient (see figure 13), its application in deep networks necessitates the usage of gradient clipping and careful weight initialization to prevent gradient explosions.

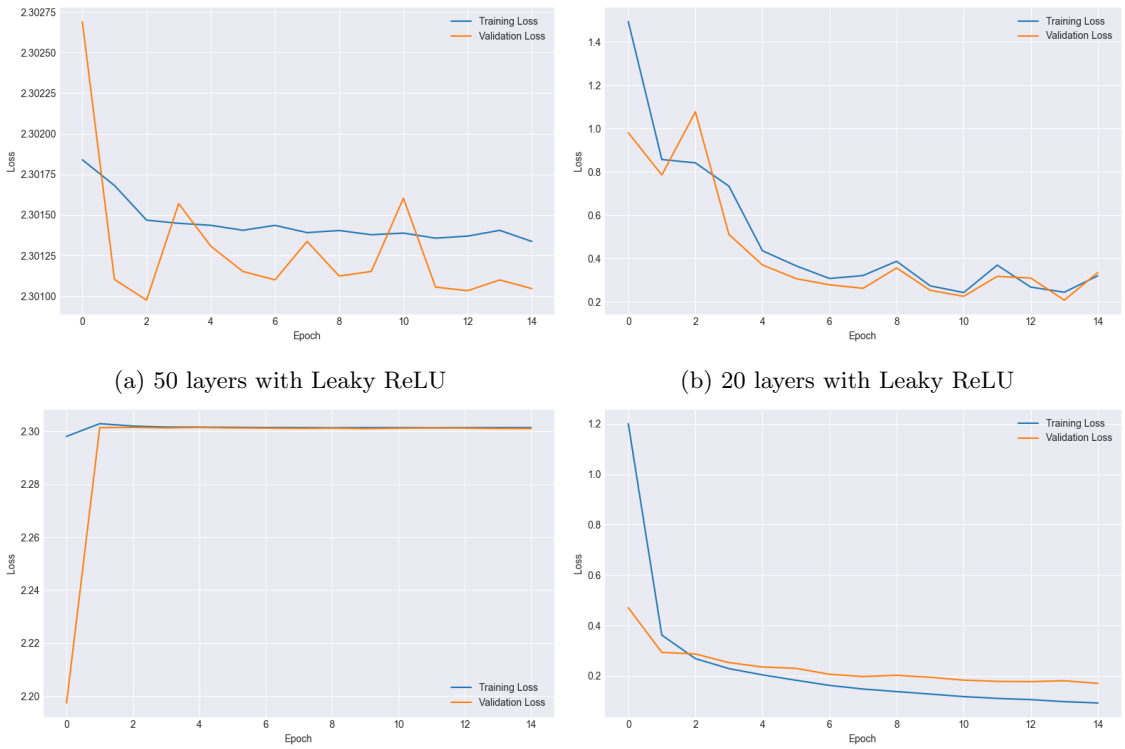
8 Exploding / Vanishing Gradients (application)

To explore how the depth of a neural network affects its ability to learn, we tested a range of fully-connected networks of different sizes. These networks were designed to classify MNIST digit images and used specific activation functions to help us understand issues like exploding and vanishing gradients. We experimented with networks as deep as 50 layers with Leaky ReLU activations to shallower ones with just 5 layers using Sigmoid activations. Each model was trained for the same duration of 15 epochs, with consistent settings for optimizers and other variables to make sure our results were comparable. Figure 14 shows how the training and validation losses changed over time for each network setup..

The experimental results depicted in the loss curves for various network architectures (see figure 14) offer revealing insights into the impact of network depth and activation functions on the training process, particularly in relation to gradient vanishing and exploding.

The deeper network with 50 layers using Leaky ReLU exhibits an initial rapid decrease in training loss, indicating effective learning during the initial epochs. However, the validation loss does not decrease as significantly, suggesting a disparity between the training and generalization capabilities. This behavior could be a symptom of the network beginning to overfit or the presence of exploding gradients that Leaky ReLU somewhat mitigates but does not fully prevent in very deep networks.

In contrast, the 20-layer network using Leaky ReLU shows a more consistent decrease in both training and validation losses, implying that at this depth, the network is capable of learning



(a) 50 layers with Leaky ReLU

(b) 20 layers with Leaky ReLU

(c) 10 layers with Sigmoid

(d) 5 layers with Sigmoid

Figure 14: Training and validation loss over 15 epochs for different network configurations.

effectively without severe overfitting or suffering from exploding gradients. The Leaky ReLU activation function, known for addressing the vanishing gradient problem by allowing a small gradient when the unit is not active, seems to maintain adequate gradient flow in networks of this depth.

Moving to the shallower architectures using the Sigmoid activation function, the 5-layer network demonstrates a steady convergence of training and validation losses. This indicates that at this depth, the Sigmoid function does not significantly hamper learning, likely because the gradients are well-preserved across fewer layers.

However, the 10-layer Sigmoid network starts to reveal the limitations of this activation function in deeper networks. The training loss decreases slowly, suggesting that the vanishing gradients problem is starting to take effect, impairing the network's ability to learn from the data effectively. The validation loss also shows signs of plateauing early, which could indicate that the network is not learning features that generalize well beyond the training data.

These results collectively highlight that while Leaky ReLU can mitigate the issue of vanishing gradients to a degree, it is not entirely immune to the problems associated with increased depth, such as overfitting or exploding gradients. On the other hand, the Sigmoid function quickly becomes a bottleneck as the network depth increases, leading to vanishing gradients and poor training dynamics. Therefore, the choice of activation function and the depth of the network must be carefully balanced to ensure efficient learning and generalization, especially when dealing with complex tasks such as digit classification on the MNIST dataset.