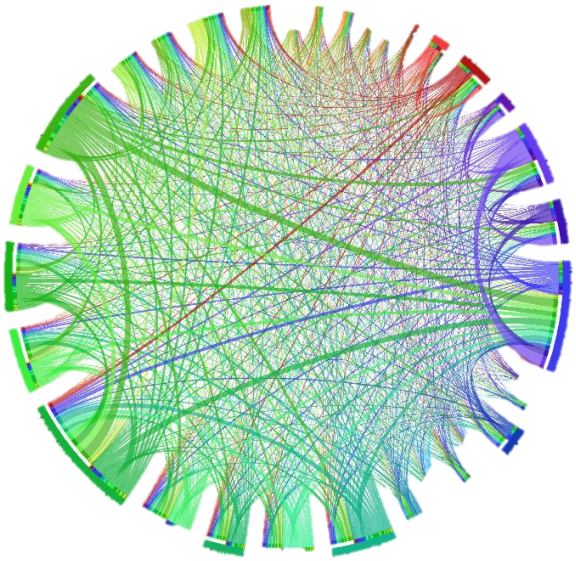
Software Engineering Department

Braude College

Citation Prediction using GAN

Final Project in Software Engineering – Course 61999



Biran Fridman

Ilya Lev

Supervisors:

Dr. Renata Avros

Prof. Zeev Volkovich

GitHub

Table of Content

Abstract

This project addresses the issue of including irrelevant citations in scholarly articles, which undermines research integrity and slows down access to reliable information and in some cases may even have a negative effect on the writers. Existing approaches to tackle this problem, such as manual screening and expert reviews do exist, but these methods suffer from subjectivity and inadequate accuracy. By utilizing Generative Adversarial Networks (GANs) and gaining insights from "A Link Prediction Algorithm Based on GAN" [1].

This project introduces a different approach. It involves creating a citation graph and utilizing the GAN-based algorithm to predict missing citations, enabling the assessment of their relevance. Successful predictions indicate probable relevance, while unsuccessful predictions suggest likely irrelevance.

1. Introduction

In today's digital age, scholarly articles play a critical role in spreading knowledge and advancing research. However, one persistent issue that researchers face is the inclusion of irrelevant citations within their articles. Irrelevant citations refer to references that do not directly contribute to the content or purpose of the article, often added for various reasons such as promoting other articles, accommodating personal relationships, or meeting external pressures.

The problem of citing irrelevant articles poses significant challenges for both authors and readers. For authors, the inclusion of irrelevant citations can undermine the integrity of their research and dilute the focus of their work. It becomes crucial for researchers to identify and exclude such citations to maintain the quality and relevance of their articles. On the other hand, readers heavily rely on citations to navigate the vast landscape of scientific literature. Irrelevant citations not only impede the readers' ability to access reliable information but also lead to time wasted in pursuing irrelevant references.

Motivation for this project arises from the need to address the persisting issue of citing irrelevant articles. While this problem has been acknowledged in the research community, finding an effective and automated solution has remained elusive. Various approaches have been explored to combat this problem, including manual screening of citations, expert review processes, and algorithmic techniques. However, these existing methods suffer from limitations such as subjectivity, time-consuming nature, and insufficient accuracy.

In the realm of link prediction, a promising technique has emerged in the form of Generative Adversarial Networks (GANs). GANs have shown remarkable success in modeling complex relationships within data, making them a suitable candidate for tackling the problem of identifying irrelevant citations. Leveraging the insights from the article "A Link Prediction Algorithm Based on GAN" [1].

The proposed solution involves adapting the GAN-based link prediction algorithm to construct a citation graph connecting articles based on their citations. By systematically removing each citation from an article and predicting the missing citation using the algorithm, the relevance of the citation to the article can be assessed. Successful prediction of the removed citation indicates

a high likelihood of relevance while unsuccessful predictions suggest a high likelihood of irrelevance, and thus highlighting citations added for reasons other than scholarly merit.

The key individuals that will benefit from this solution are researchers, authors, academic institutions, publishers, and readers.

In this research project, we aim to help academic researchers tackle the problem of citing irrelevant articles. Our goal is to improve the quality and relevance of scientific literature and make it easier for researchers to find valuable information. By using a special algorithm based on GANs, we can analyze the citations in articles and identify those that may not be relevant. This will save researchers time and effort by streamlining the citation process and ensuring that they only include citations that truly contribute to their work.

2. Background and Related Work

2.1 Link Prediction

Link prediction is a technique used in network analysis to predict missing links or future connections between nodes in a network. It's based on the assumption that the probability of a link between two nodes depends on the similarity of their attributes or their connectivity patterns with other nodes in the network.

In other words, link prediction aims to identify pairs of nodes that are likely to be connected in the future or that have a high probability of being connected but are currently not connected in the network.

2.2 Principal Component Analysis (PCA)

PCA is a statistical technique used for dimensionality reduction. It simplifies the complexity in high-dimensional data by transforming it into a new set of variables, called principal components, which are uncorrelated and ordered so that the first few retain most of the variation present in all of the original variables. PCA helps in emphasizing variation and bringing out strong patterns in a dataset, enabling data to be visualized and analyzed more effectively.

2.3 Communities (Clusters)

Communities represent groupings of elements in a dataset that share common characteristics or features, making them more similar to each other than to elements outside their group.

In the context of link prediction, identifying communities can be pivotal. For example, in a citation network, clusters may consist of research papers that cite each other frequently, indicating shared subject matter or methodologies. Clustering helps to simplify the complexity of networks by highlighting these natural groupings, which can then inform predictions about which new links are likely to form, based on the principle that new connections are more probable within the same community.

In this project we use the Community detection algorithm that Finds the community structure of the network according to the Infomap method of Martin Rosvall and Carl T. Bergstrom [9].

2.4 Cora Dataset

The Cora dataset consists of 2708 scientific publications classified into one of seven classes. The citation network consists of 5429 links. Each publication in the dataset is described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary. The dictionary consists of 1433 unique words that are most common in those fields.

2.5 Adam Optimizer

The Adam optimizer, standing for Adaptive Moment Estimation, is a popular algorithm in the field of deep learning for its efficient computation of adaptive learning rates for each parameter. By combining the benefits of two other extensions of stochastic gradient descent, namely AdaGrad and RMSProp, Adam adjusts the learning rate throughout training, providing an effective solution for optimizing neural networks.

2.5.1 RAdam Optimizer

Building upon Adam, the Rectified Adam (RAdam) Optimizer introduces a rectification mechanism to address Adam's variance and convergence challenges during the initial training phase. This rectification term dynamically adjusts the learning rate, mitigating the premature variance adaptation problem. The key advantage of RAdam over traditional Adam is its ability to offer a more stable and consistent optimization, leading to faster convergence and improved model performance without the necessity for extensive hyperparameter fine-tuning. RAdam's approach to rectifying the adaptive learning rate's variance ensures a smoother training process, making it a robust choice for complex machine learning models.

2.6 Loss Functions

Loss functions are crucial in the training of machine learning models, providing a measure of how well the model's predictions match the actual data. They guide the optimization process, with the objective of minimizing the loss to improve model accuracy.

2.6.1 Loss Function: MSE

Mean Squared Error (MSE) is a loss function commonly used for regression tasks. It calculates the average of the squares of the differences between the predicted values and the actual values. This approach to error measurement places a greater penalty on larger errors, making it sensitive to outliers in the data set. MSE is particularly useful when it is important to emphasize larger errors more significantly than smaller ones, which can be crucial in many machine learning models where accuracy in predictions of higher values is desired. The squaring of errors inherently gives a higher weight to larger discrepancies, making MSE a valuable tool for identifying and correcting predictions that are significantly off from actual values.

2.6.2 Loss Function: BCE

Binary Cross-Entropy (BCE), on the other hand, is commonly used for binary classification problems. It measures the distance between two probability distributions - the predicted probability distribution and the actual distribution, where the actual outcomes are either 1 or 0. BCE quantifies the performance of a classification model whose output is a probability value between 0 and 1. This loss function is especially effective for models where the prediction of the probability for

each class is crucial, as it penalizes incorrect classifications more heavily when the model is more confident about its prediction.

2.7 Cosine Similarity

Cosine similarity measures the cosine of the angle between two vectors in a multidimensional space, ranging from -1 (dissimilar) to 1 (similar). In graph theory, it assigns edge weights by quantifying the similarity between nodes. For example, in a citation network, cosine similarity between document vectors can determine the weight of edges, reflecting the content similarity of connected documents. This approach enhances the analysis of relationships within graphs, useful for link prediction and community detection.

2.8 Complex network Graph

A network graph $G = (V, E)$ consisting of a Vertex set V and an Edge set E, where each vertex represents a data object, and each edge represents a relationship between the data objects. For a given vertex V_c, $N(V_c)$ represents the vertex directly connected to V_c, the direct neighbor of V_c.

2.9 Generative countermeasure network (GAN)

GAN is a machine learning model framework that consists of two neural networks, namely the **Generator** and the **Discriminator**. GANs are designed to generate realistic data samples that resemble a training dataset such that the discriminator could not determine between synthetic data and real data.

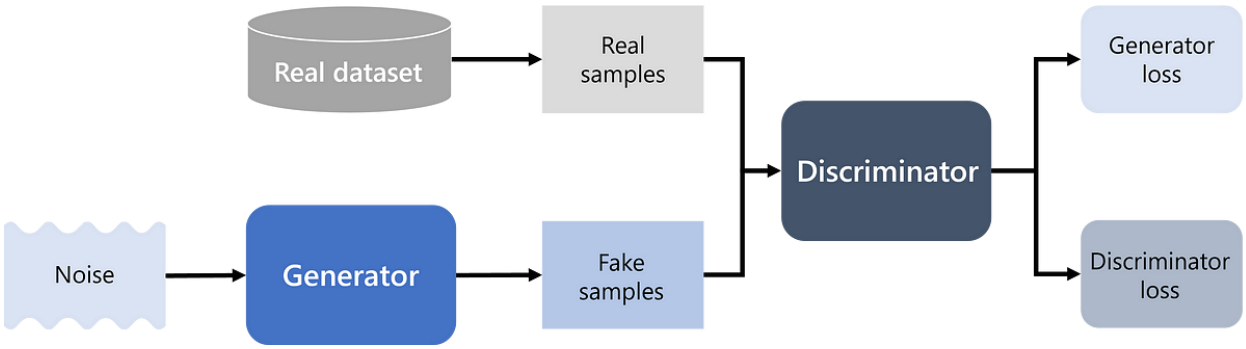


Figure 1. Example of GAN network structure [7].

The **Generator** is the part of the network responsible for generating synthesized data based on the real data. At each iteration, the generated synthesized data is passed to the discriminator, and based on the feedback from the discriminator, the generator makes correction to generate data as similar to the real data as possible such that the discriminator cannot distinguish between real data and synthesized data.

The **Discriminator** is the second part of the network, trained to distinguish between real data samples from the training set and the synthetic samples produced by the generator. It aims to

correctly classify whether a given input is real or generated. The discriminator is trained using a combination of real and generated data samples, adjusting its parameters to improve its ability to differentiate between the two, while passing the results to the generator.

During the training process, the generator and discriminator networks are pitted against each other in a game-like manner. The generator aims to generate more realistic samples to fool the discriminator, while the discriminator aims to become better at distinguishing between real and generated data. The networks are trained iteratively, with the generator trying to minimize the discriminator's ability to differentiate, and the discriminator trying to maximize its discriminative power.

The training process of a GAN involves a back-and-forth competition between the generator and discriminator networks. As the training progresses, the generator becomes better at generating realistic samples, while the discriminator becomes more skilled at identifying generated data.

This iterative process continues until a balance is reached, ideally resulting in a generator that can produce high-quality synthetic samples that are indistinguishable from real data.

In the contexts of link prediction, a GAN is a type of neural network that can be used to predict missing links in a network.

The goal of a GAN model is to generate new links in a network that are likely to be present and are currently missing.

2.10 Hierarchical Network

The hierarchical processing of a network graph refers to the approach of analyzing and understanding the graph's structure and information in a hierarchical or multi-level manner. It involves breaking down the graph into different levels or layers and performing computations and analyses at each level to gain insights into the network's organization and properties.

In a hierarchical processing framework, the network graph is represented as a collection of nodes and edges, where nodes represent entities, and edges represent relationships or connections between entities. The goal is to understand the graph's characteristics and connectivity patterns by considering different levels of abstraction.

At each level of the hierarchy, various techniques and algorithms can be employed to extract meaningful information. This hierarchical analysis helps to reveal the complex organization and relationships within the network [1].

2.11 Node Embedding

Node embedding is a way of representing nodes as low – dimensional vector representations. The goal of node embedding is to capture the structural and semantic information of nodes in a way that preserves their relationships and properties within the graph.

2.11.1 Random walk

Random walk is a mathematical model used to describe a random process that involves moving through a network or graph, where each step is taken at random based on a set of probabilities. In

a random walk, an entity starts at a particular node in a network and takes a step to a neighboring node at random, with each neighboring node having a certain probability of being chosen as the next step. The process then continues, with the entity taking successive steps at random until it reaches a stopping condition.

Random walks are not completely random, the way to control the walk is by giving weight for the edges in the graph.

The weight is determined by the function $\alpha(t, V_i)$ where t is the previously visited node, V_i is the potential next node and d_{tvi} is the distance (shortest path) from note t to the following node V_i.

The p and q parameters are used to control the probabilities of moving towards neighboring nodes during a random walk. Specifically, p determines the likelihood of returning to the previous node in the walk, while q determines the likelihood of exploring a new, unvisited node. When p is low and q is high, the random walk favors exploring nodes that are far away from the starting node, resulting in a "depth-first search" behavior. Conversely, when p is high and q is low, the walk favors exploring nodes that are close to the starting node, resulting in a "breadth-first search" behavior. By tuning these parameters, the algorithm can capture different types of structural information in the network [5].

$$\alpha(t, v_i) = \begin{cases} \frac{1}{p}, d_{tv_i} = 0 \\ 1, d_{tv_i} = 1 \\ \frac{1}{q}, d_{tv_i} = 2 \end{cases}$$

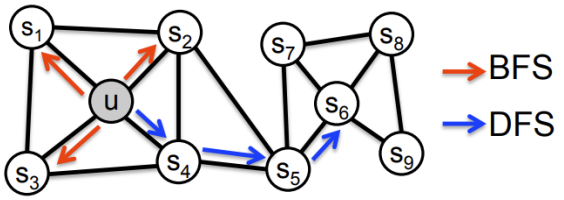


Figure 2. Example of BFS and DFS Algorithms [8].

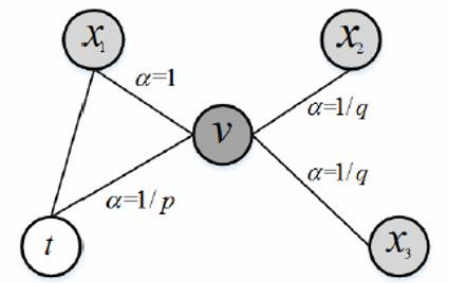


Figure 3. Settings of deviation coefficient in random walk [5].

2.11.2 Node2Vec Algorithm

Node2Vec is an algorithm used to create vector representations of nodes in a graph for machine learning applications. It works by using random walks through the graph starting at a target node to learn low-dimensional representations of each node. The algorithm treats the random walks like sentences in a corpus, where each node is treated as an individual word and a random walk is treated as a sentence. By feeding these "sentences" into a skip-gram or continuous bag of words model, the paths found by random walks can be treated as sentences and traditional data-mining techniques can be used. Node2vec is an improvement on prior work that was based on rigid notions of network neighborhoods and argues that the added flexibility in exploring neighborhoods is the key to learning richer representations of nodes in graphs. It is considered one of the best graph classifiers [2].

3. Solution – GAHNRL Algorithm

The project aims to address the issue of irrelevant citations in scholarly articles by using a GAN-based algorithm. The process involves creating a citation graph from the Cora dataset, applying machine learning techniques to predict the relevance of citations, and thus, identifying probable relevant or irrelevant citations.

- Using the network graph layering algorithm explained in 3.2 to create a subnetwork of n graphs.
- Process graph G_n using Node2Vec Algorithm explained in 2.11.2 to generate the initial vector representation of the vertices of the graph G_n.
- The initial vector representations of each layer in the subnetwork graph are recursively fed into the generative countermeasure network for training, starting from graph G_n.
- The algorithm employs the EmbedGAN algorithm explained in 3.5 to learn low-dimensional vector representations of vertices in G_n. These learned representations serve as the initial vector representations for the upper layer subnetwork graph, G_{n-1}.

  The process recursively continues through backtracking learning until reaching the initial network graph, G_0. Ultimately, the algorithm achieves the low-dimensional vector representation, E_{G_0}, for all vertices.

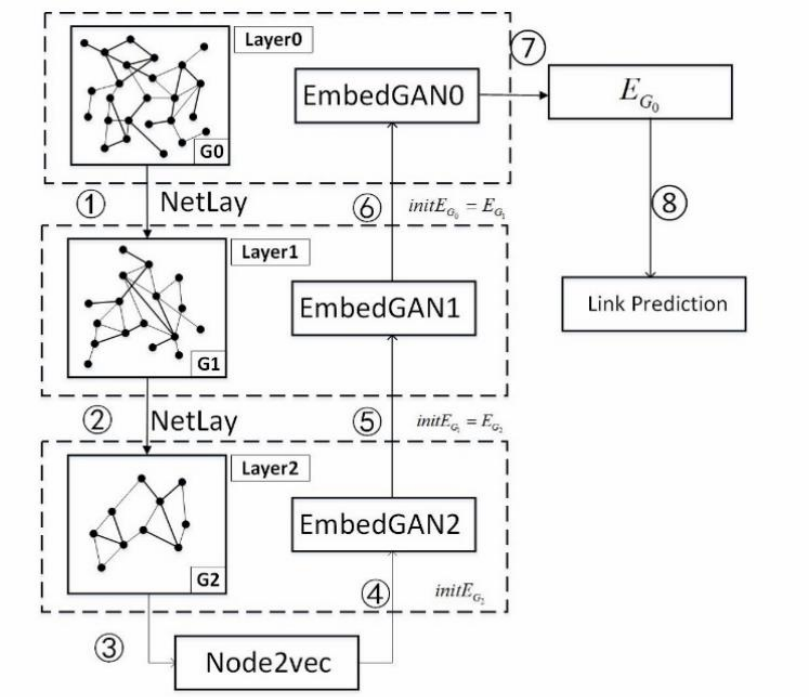


Figure 4. Framework of the algorithm [1].

3.1 Data Pre-Processing

This stage initiates the transformation of raw data into a structured format conducive to machine learning analysis, encompassing both the construction of the citation network and the refinement of node features.

3.1.1 Graph Construction

Begins with the creation of a directed graph derived from the Cora dataset. Each node represents a scholarly article, while edges signify citations between these articles, thereby establishing a citation network that mirrors the scholarly discourse captured in the dataset.

3.1.2 Edge Weight Assignment

To encapsulate the relationship strength between cited works, we assign weights to the edges of the graph. These weights are computed using the cosine similarity between the word vectors of the citing and cited articles. This step ensures that the graph captures not just the citation links but also the content similarity between connected articles.

3.1.3 Feature Vector Resizing

Given the high dimensionality of the word vectors obtained from the dataset (1433 dimension), we employ Principal Component Analysis (PCA) to reduce these vectors to a dimensionality that matches the intended size of the node embeddings.

3.2 Hierarchical Network layering algorithm – Lev - Fridman Netlay

This phase focuses on simplifying the complex citation network into a series of progressively coarser graphs. Through hierarchical network layering, we aim to distill the essential structure of the network, making it more manageable for analysis and facilitating more accurate predictions of citation relevance.

The LF_NetLay Algorithm is central to the hierarchical layering process, designed to methodically reduce the graph's complexity while preserving its fundamental connectivity and community structure.

3.2.1 Identifying Communities

The initial step involves detecting communities within the graph, grouping articles that exhibit a high degree of interconnectedness due to similar citations. This clustering sheds light on the network's inherent structure, allowing us to pinpoint areas of dense scholarly communication.

we utilize the Community detection algorithm that Finds the community structure of the network according to the Infomap method of Martin Rosvall and Carl T. Bergstrom [9].

3.2.2 Graph Coarsening

Merges nodes within the same community into super-nodes, effectively reducing the graph's size. This process also involves creating new edges between these super-nodes when there exists any edge connecting nodes of the respective communities in the original graph. The weight of the new edge is the aggregated weight of all such connecting edges, ensuring that the new, simplified graph retains the original network's connectivity patterns. In addition, the algorithm doesn’t take edges between nodes from the original graph that belong to the same cluster in the new graph, The initial step involves detecting communities within the graph, grouping articles that exhibit a high degree of interconnectedness due to similar citations. This clustering sheds light on the network's inherent structure, allowing us to pinpoint areas of dense scholarly communication.

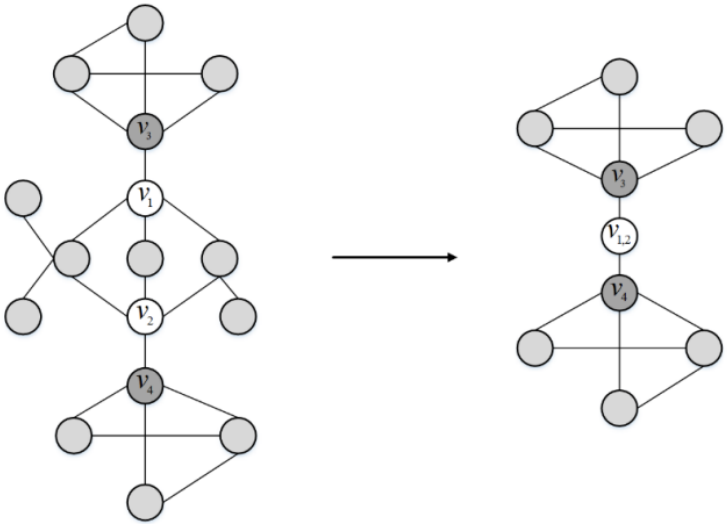


Figure 5. Example of network graph layering [1].

3.2.3 Normalizing Edge Weights

the final step involves the normalization of edge weights in the newly simplified graph. By scaling these weights to a uniform range, this helps facilitating more straightforward analysis and interpretation in later stages of the project.

3.3 Embedding Generation using Node2Vec

In this crucial phase, we leverage the Node2Vec algorithm to generate embeddings for each node within the most simplified layer of our hierarchical graph. Node2Vec, adept at capturing both the structural and feature-based nuances of nodes, is applied to this coarsened graph, enabling a nuanced representation of scholarly articles as dense vector embeddings. These embeddings encapsulate the intricate web of citations and thematic similarities into a form that is both computationally efficient and rich in information.

The process begins by utilizing Node2Vec on this last iteration graph—the version that represents the essence of the original citation network. By walking through the graph in a manner that balances local fidelity and global contextuality, Node2Vec translates the connectivity patterns and the inherent properties of nodes into a vector space. This transformation results in each node being represented by a fixed-size vector, capturing its positional and relational characteristics within the network.

Following the generation of node embeddings, we proceed to construct what we refer to as positive sample embeddings. For every pair of directly connected nodes where each pair representing an actual citation link in the simplified graph—we concatenate the embeddings of the two nodes to form a representation of that edge. This concatenation acts as a bridge, merging the distinct information carried by each node into a unified vector that embodies the connection's characteristics. These positive sample embeddings play a pivotal role in both the pre-training and subsequent training phases, serving as authentic examples against which the model learns to gauge the veracity of generated embeddings.

By meticulously crafting these embeddings, we lay a foundational bedrock for the GAN model's learning process. The positive samples, derived from the actual citation links, provide a ground truth of sorts, enabling the model to understand and replicate the complex patterns of scholarly citations. This initial step not only prepares the model for effective pre-training but also ensures that the subsequent phases of model training are grounded in an accurate and insightful representation of the citation network's structure and dynamics.

3.4 Pre-Training

Following the generation of embeddings and positive samples, this section will delve into the pre-training phase for both the generator and discriminator components of the GAN model. It will describe how the positive samples serve as a foundational dataset for this pre-training, enabling both model components to learn the initial distinctions between genuine and artificially generated embeddings. This phase is crucial for establishing a baseline understanding within the model, from

which it can refine its ability to generate and evaluate embeddings that accurately reflect plausible links in the citation network.

3.4.1 Generator Pre-Training

In the generator pre-training phase, we focus on training the generator to produce node embeddings that closely resemble those of actual neighbor pairs in the graph. Utilizing the RAdam optimizer for its efficiency and Mean Squared Error (MSE) for its sensitivity to the average magnitude of errors, this stage fine-tunes the generator through a series of epochs. The process involves generating synthetic embeddings from noise vectors, comparing these with real neighbor pairs using MSE, and adjusting the generator's parameters based on this comparison. Early stopping is implemented to halt training when improvements fall below a specified threshold, ensuring that the generator does not overfit to the training data. This careful calibration of the generator's ability to mimic real embeddings is crucial for the model's overall performance in predicting plausible citation links.

3.4.2 Discriminator Pre-Training

Following the generator's pre-training, the discriminator undergoes a similar preparatory phase. Its role—to discern between real and artificially generated embeddings—requires precision that this training phase aims to instill. The RAdam optimizer and BCE (Binary Cross-Entropy) loss function are chosen for their effectiveness in binary classification tasks, such as distinguishing between genuine and fake embeddings. By training on a combination of real neighbor pairs and fake embeddings produced by the generator, and adjusting parameters based on the BCE loss, the discriminator learns to accurately evaluate the authenticity of embeddings. Early stopping here too ensures that training ceases when no further significant improvements are detected, preventing overfitting, and setting the stage for a discriminator that adeptly identifies embeddings reflective of true citation relationships.

3.5 Recursive EmbedGAN Training Process

This section delves into the iterative approach of traversing through the layers of the hierarchical network, utilizing the EmbedGAN algorithm at each layer to understand its connections and properties while also evaluating the model's performance. Starting from the most simplified layer, the process not only trains the model to recognize and replicate the network's structural nuances but also assesses its predictive accuracy at each step. This recursive training is crucial for adapting the model's understanding and effectiveness across different levels of abstraction within the citation network.

3.5.1 Builder Sampling Strategy

The Builder Sampling Strategy is the initial step in preparing the network for the EmbedGAN training. By applying the BiasedRandomWalk on the network graph, we generate walks that reflect the underlying structure and connectivity of the graph. These walks, filtered to include only valid paths that are present in the directed graph, serve as the basis for creating positive samples. The maximum path length for these walks is determined by calculating the average shortest path length across all components of the graph, ensuring that the walks are representative of the actual

distances within the network. This strategic sampling lays the groundwork for effective model training by providing a rich dataset that encapsulates the network's complexity.

3.5.2 Training the GAN Model

The GAN training process is where the core of the EmbedGAN training unfolds. Utilizing the generated walks and the node embeddings, the function systematically trains both the generator and discriminator components through a specified number of epochs and batch sizes. The training involves alternating between optimizing the discriminator to distinguish between real and generated data, and refining the generator to produce increasingly accurate embeddings. Notably, the process incorporates a K-Fold mechanism to enhance the robustness of the training by validating across multiple subsets of the data. The iterative training and optimization are designed to steadily improve the model's ability to predict plausible links, with the evaluation of precision, recall, and F1 score reserved for a dedicated evaluation phase.

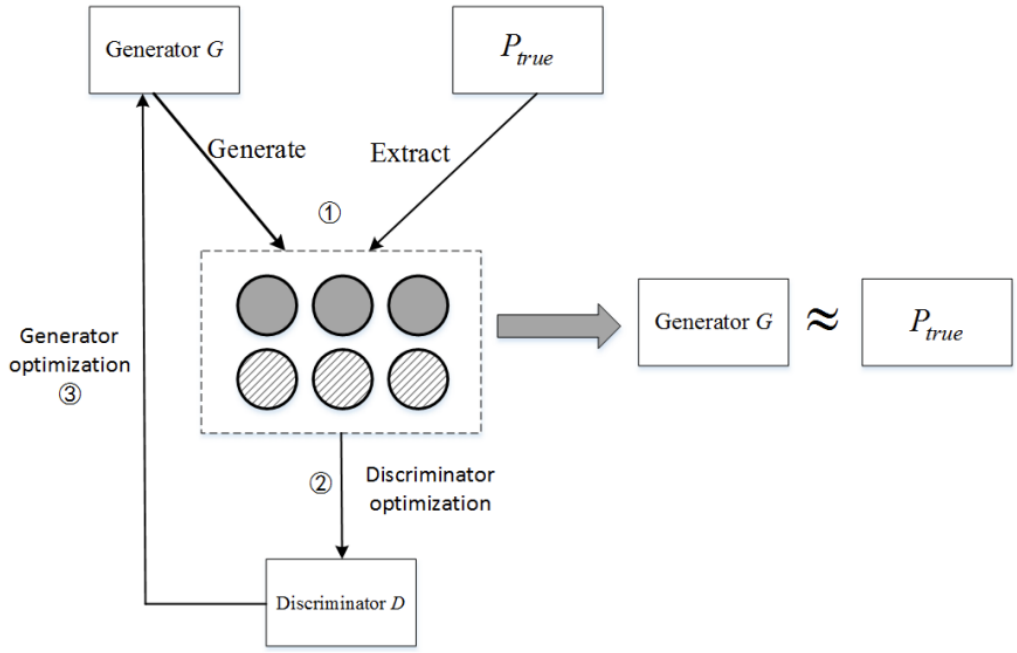


Figure 6. EmbedGAN Algorithm Framework [1]

3.5.3 Assignment of Embeddings for the Next Layer

Following the training and evaluation within each iteration, the process includes a crucial step of assigning updated embeddings to nodes for the subsequent layer. This assignment involves integrating community embeddings with individual variations based on the node's feature vectors and the community to which it belongs. By introducing unique variations for each node, the method ensures that the embeddings reflect not only the collective characteristics of their communities but also the distinctive attributes of each node. This tailored approach to updating embeddings is essential for maintaining the model's adaptability and accuracy as it progresses through the hierarchical layers of the network.

3.6 Final Link Predictions

This final section outlines the process for predicting the relevance of links within the citation network, utilizing the trained discriminator model to assess the potential relevance of each citation link based on their embeddings. The aim is to determine which edges (citations) are relevant and contribute meaningfully to the scholarly dialogue and which do not.

3.6.1 Embedding Generation for Edges

Initially, we generate embeddings for all edges in the graph by concatenating the embeddings of the nodes that form each edge. This step converts the abstract representation of node relationships into a format that can be directly analyzed by the discriminator.

3.6.2 Making Final Predictions

We randomly select a predetermined percentage (e.g., 30%) of these edge embeddings to undergo evaluation. The selected embeddings are then fed into the discriminator, which outputs a prediction score for each one. These scores are between 0 and 1, where scores closer to 1 suggest a higher relevance of the citation link.

These scores are then converted into binary predictions using a threshold (commonly 0.5), where scores above this threshold are considered predictions of relevant links.

3.6.3 Evaluating Predictions

The binary predictions are used to determine the number of relevant and irrelevant connections identified by the model.

Statistics such as the number of relevant and irrelevant connections and the success rate (ratio of relevant connections to total assessed connections) are calculated and reported. These metrics provide insights into the model's effectiveness and accuracy in identifying relevant citations.

4. Development Process

Our project was inspired by the article "A Link Prediction Algorithm Based on GAN", which discusses a method for link prediction emphasizing GAN architecture.

Guided by our academic advisors, we tailored the approach outlined in the original paper to meet our specific objective: identifying relevant citations within academic articles. This adaptation marked the beginning of our work, detailed in Part A of this book, where we explore the theoretical foundation for our problem using methods from the cited article.

Initially, we attempted to locate the code repository of the original authors by searching on GitHub using the article's title and the authors' names, but to no avail. Efforts to contact the authors through various social platforms like LinkedIn and Google Collab were also unsuccessful. Consequently, we turned to alternative resources, such as the 'Papers with Code' website. However, this approach was somewhat limited due to the novelty of using GAN networks and citation networks for link prediction.

Determined to advance our understanding, we utilized resources such as YouTube and OpenAI's ChatGPT. Ilya, striving to deepen his expertise, enrolled in every relevant machine learning and

AI course available during his final semester. Balancing part-time jobs as software developers and pursuing a bachelor's degree in social engineering, we found time to be a scarce resource.

With limited time and knowledge, we committed to maximizing every available moment, breaking down our tasks into smaller, manageable components. Progress was gradual but steady, as our knowledge and codebase expanded. We conducted coding sessions together due to the complexity of the topic, which was challenging given our initial lack of deep technical knowledge. Independently, we dedicated spare moments to further our understanding through reading and watching educational content, even using time between commits to resolve emerging issues in our development process.

Initially, we used Google Collab for our development work, but due to significant obstacles that we will discuss in the following chapter, we eventually transitioned to working on personal computers at home. Given that our hardware was not particularly powerful, this too was a time-consuming adjustment.

5. Tools and Resources

Throughout the development of our project, we utilized a variety of tools and resources, which were instrumental in the successful execution of our citation prediction system:

1. **NumPy Documentation**: An essential resource for numerical computing in Python. NumPy Documentation
2. **StellarGraph API**: Provided tools for machine learning on graphs. StellarGraph API Documentation
3. **iGraph API**: Used for creating and manipulating undirected and directed graphs. iGraph Documentation
4. **Scikit-learn**: A machine learning library for Python, used for implementing regression, classification, and clustering algorithms. Scikit-learn Documentation

Initially, we used Google Collab for its ease of use and accessibility. However, we encountered compatibility issues with Google Collab's Python version (3.10), which necessitated a switch to a local environment. We opted for Python 3.8 and used Visual Studio Code as our primary development environment, running our code on personal machines to better manage our specific dependencies.

Additionally, we frequently referred to Papers with Code, a valuable online resource that helped us identify relevant algorithms and techniques. This was particularly useful as it supplemented our initial lack of expertise in the specific areas required for this project.

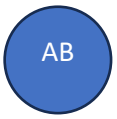
6. Challenges and Solutions

6.1 Challenge: Graph Directionality in Citation Networks

Unlike social networks that are inherently undirected, citation networks are directional. This posed a significant challenge when attempting to adapt methods from the foundational paper, which were originally designed for undirected graphs like those representing social connections. For example, in social networks, a connection (friendship) between two nodes (people) is bidirectional by nature. However, in citation networks, if article 'A' cites article 'B', it does not imply that 'B' also cites 'A'.

This discrepancy necessitated a re-evaluation of the hierarchical graph techniques proposed in the original paper, specifically 'edge folding' and 'vertex merging'. These methods assume undirected edges, which simplifies the process of merging nodes without considering the direction of influence. In our citation graph, merging nodes without considering citation direction would incorrectly suggest mutual citations between papers, distorting the scholarly context.

To adapt these techniques to a directed graph, we had to modify the edge folding process to create a super node (e.g., combining articles A and B into AB) in a manner that preserves the directionality, ensuring that the resultant node accurately represents the flow of citations. For vertex merging, the challenge was more pronounced because it could potentially reverse the direction of citations, which is unacceptable in our context.



Solution: Infomap Method for Community Detection

After realizing the limitations of the original hierarchical methods for directed graphs, we explored alternative approaches and discovered the Infomap method, detailed in "Maps of Random Walks on Complex Networks Reveal Community Structure" by Martin Rosvall and Carl T. Bergstrom. Infomap leverages the flow of random walks to detect community structures, making it highly effective for directed networks like ours. It interprets real-world traffic flows and traces paths through the network to reveal clusters of articles that cite each other more frequently, thus maintaining the correct citation direction.

6.2 Challenge: Software and Environment Compatibility Issues

Initially, we utilized Google Collab for its pre-configured environment, which facilitated easy access to powerful computing resources. However, we encountered a critical issue with Python version compatibility. The Biased Random Walk algorithm, essential for our model, required

Python versions <=3.9, whereas Google Collab supported Python 3.10. Efforts to work around this by configuring a virtual environment within Collab or adapting the code to function under Python 3.10 were unsuccessful.

Solution: Transition to Local Development Environment

To resolve the version incompatibility, we decided to migrate our development environment to local machines. This involved setting up Python 3.8, which was compatible with all our dependencies. The transition required downloading all necessary libraries, configuring the environment, and adapting our codebase for local execution. Although this shift was resource-intensive—requiring substantial time to configure and a week to fully transition—it allowed us to continue our development without further compatibility issues.

6.3 Challenge: Computational Limitations of Personal Hardware

Once we shifted to local machines, another challenge emerged: the inferior computing power of our personal hardware compared to Google Collab's cloud-based resources. This limitation was particularly impactful because the complex GAN-based models required extensive computational resources, leading to lengthy training times of up to ten hours per session.

Solution: Optimization and Strategic Development

To mitigate the impact of limited hardware, we optimized our code to reduce runtime and carefully planned each modification to the model. We adopted a more theoretical approach to changes, predicting their impacts before implementation to minimize unnecessary computations. This strategy not only conserved computing resources but also deepened our understanding of the model's dynamics, allowing for more thoughtful and effective iterations.

6.4 Challenge: Conceptualizing and Implementing Link Prediction

One of the most intellectually demanding challenges was conceptualizing how to mathematically represent link prediction within our model. We needed a method to determine, from the embeddings alone, whether one article should cite another.

Solution: Vector Concatenation and GAN Training

We addressed this by proposing to concatenate the embeddings of two articles, transforming them into a single embedding that our GAN could process. By training the generator to recreate these concatenated embeddings and teaching the discriminator to differentiate between real and synthetic examples, we were able to implement a robust link prediction mechanism.

6.5 Challenge: Discrepancies in Node Embeddings Across Graph Layers

As we progressed in the implementation of our citation prediction algorithm, a critical challenge emerged during the final stages—specifically when about three-quarters of the way through. After successfully applying a hierarchical breakdown to our graph, we generated embeddings for the nodes of the most abstracted graph, denoted as G_n . This graph G_n being the most abstract representation of our original graph G, contained the smallest number of vertices V and edges E.

Technical Explanation of the Issue:

When decomposing the initial graph G into various levels of abstraction defined as G_i (where i represents the current level of abstraction), G_n is the final and most simplified form of G. At this abstraction level, let's say $|V| = n$, we applied the Node2Vec algorithm to generate embeddings for each node within G_n These embeddings were crucial for the machine learning tasks that followed—specifically, training and testing our model based on the graph structure at this level of abstraction.

However, the challenge arose when examining the graph at the previous abstraction level, G_{n-1}, which contained $n + k$ nodes. This discrepancy posed a significant problem: we had only n embeddings from G_n but needed to account for $n + k$ nodes in G_{n-1}.

Initial Considerations and Innovative Solution:

Initially, we were somewhat perplexed by how to generate the additional k embeddings required for the nodes in G_{n-1}. Our academic advisors proposed a straightforward solution—assigning the same embedding to nodes that were grouped into the same super node during the abstraction process. However, upon deeper reflection, we realized that this approach might oversimplify the complex inter-node relationships and potentially obscure unique characteristics of individual nodes within each super node.

Given that we were utilizing the Cora dataset, which describes each publication with a binary vector representing the presence or absence of specific words (from a dictionary of 1433 unique words), we identified an opportunity to enrich our node embeddings. We decided to leverage this rich metadata to introduce variability into the node embeddings across different graph layers.

Solution:

To implement this, we took the existing embedding for each super node cluster from G_n and introduced variations for individual nodes within the cluster. Utilizing Principal Component Analysis (PCA), we transformed the high-dimensional binary vectors (1433-dimensional) into a more manageable size (128-dimensional), which we referred to as the 'features' of each node. By combining the general embedding of the super node with these individualized features, we generated distinct embeddings for each node in G_{n-1}. This approach allowed us to create the necessary $n + k$ embeddings, thereby maintaining the integrity and granularity of our model's input data across different levels of graph abstraction.

7. Results and Conclusions

7.1 Project Goals and Achievements

Our project set out with the primary goal of utilizing a Generative Adversarial Network (GAN) to predict relevant and irrelevant citations in scholarly articles. By creating a citation graph from the Cora dataset and implementing the GAN-based link prediction algorithm, we aimed to systematically predict and evaluate citation relevance.

After 30 iterations of model training and testing, our results, meticulously recorded in the Excel sheet, reveal a compelling achievement of this goal. Each edge's success in the graph across these

iterations was quantified, showing how frequently the model accurately predicted the link between nodes.

7.2 Analysis of Results

We classified the success rates into five categories to provide a structured interpretation of our model's performance:

- Relevant Citations: Success rate of 100%
- High Likelihood of Relevant Citation: Success rates between 75% and 99%
- Lower Likelihood of Relevant Citation: Success rates between 50% and 74%
- High Likelihood of Irrelevant Citation: Success rate between 25% and 49%
- Irrelevant Citation: Success rate below 25%

Examples for Each Category:

1. Relevant Citations: Node pair (1152711, 241821) from the field of Neural Networks displayed a perfect success rate across iterations, underscoring strong relevance.

Source Node	Destination Node	Source Field	Destination Field	Sampled Amount	Success Rate
1152711	241821	Neural Networks	Neural Networks	12	100.00%

Figure 7. An example of an relevant citation.

2. High Likelihood of Relevant Citation: Node pair (1112686, 18811), both in Probabilistic Methods, had a success rate of 88.89%, indicating a high probability of relevance.

Source Node	Destination Node	Source Field	Destination Field	Sampled Amount	Success Rate
1112686	18811	Probabilistic Methods	Probabilistic Methods	9	88.89%

Figure 8. An example of an High Likelihood relevant citation.

3. Lower Likelihood of Relevant Citation: Nodes (1108175, 18781) one from Neural Networks field and the other from Theory recorded a success rate of 62.50%, suggesting a moderate connection.

Source Node	Destination Node	Source Field	Destination Field	Sampled Amount	Success Rate
1108175	18781	Neural Networks	Theory	8	62.50%

Figure 9. An example of an Lower Likelihood relevant citation.

4. High Likelihood of Irrelevant Citation: An example from this category includes node pair (1154525, 1213) with a success rate of 30%, indicating a probable lack of relevance.

Source Node	Destination Node	Source Field	Destination Field	Sampled Amount	Success Rate
17208	36131	Reinforcement Learning	Probabilistic Methods	7	28.57%

Figure 10. An example of an High Likelihood of Irrelevant citation.

5. Irrelevant Citation: The interaction between nodes (1128430, 3220) one from case based field and the other from Neural Networks, showed a dismal success rate of 0%, highlighting their irrelevance.

Source Node	Destination Node	Source Field	Destination Field	Sampled Amount	Success Rate
1128430	3220	Case Based	Neural Networks	6	0.00%

Figure 11. An example of an Irrelevant Citation.

These results not only validate the effectiveness of our model but also illustrate the potential for automating citation validation processes, enhancing the integrity and efficiency of scholarly communication.

7.3 Addressing Challenges: Fine-Tuning Model Parameters

Fine-tuning the Generative Adversarial Network (GAN) for our project presented significant challenges due to the sensitivity of the model to its configuration parameters. The complexity of tuning elements like the loss functions, optimizers, learning rates, and the number of epochs required a highly iterative and time-consuming approach.

7.3.1 Key Parameters and Their Impact

Selecting the right optimizer and loss functions was critical. We opted for the Rectified Adam (RAdam) optimizer, known for its effectiveness in handling sparse gradients and improving convergence. The generator's performance was particularly sensitive to the choice of the Mean Squared Error (MSE) loss function and the same applies to the discriminator's BCE loss function, which both needed precise adjustment to accurately reflect prediction errors.

7.3.2 Computational Demands and Iterative Testing

Each minor adjustment in parameters like learning rates or K-fold numbers necessitated a full day of model retraining. This was essential to observe the direct impact of changes on the model's performance, given our limited computational resources. The iterative testing process was not only time-intensive but also required strategic planning to maximize resource utilization.

7.3.3 Strategic Decision-Making

Balancing theoretical knowledge, empirical testing, and practical constraints was crucial. We often had to predict the potential impacts of parameter changes before actual implementation. This strategy helped manage our resources efficiently but also posed risks of missing optimal settings, which we mitigated through continuous refinement.

7.3.4 Addressing Challenges Conclusion

The fine-tuning process underscored the intricate balance required between model complexity and computational feasibility, emphasizing the need for a deep understanding of machine learning model training within practical constraints.

7.4 Considerations in Decision-Making

Our decision-making process was guided by both the theoretical frameworks established in the literature and practical constraints encountered during implementation. The choice to perform 30 iterations, for example, was informed by a need to statistically validate our model's predictions while remaining within the limits of our computational resources.

Furthermore, the analysis of node fields added a nuanced layer to our evaluation, allowing us to understand not just the existence but also the contextual relevance of citations based on the similarity of research areas.

7.5 Conclusion

The project successfully demonstrated the application of a GAN-based algorithm in predicting citation relevance with a high degree of accuracy. This approach offers a promising avenue for enhancing the quality of scholarly databases by filtering out irrelevant citations, thereby streamlining the research process and ensuring the credibility of academic publications.

In conclusion, while our results are encouraging, continuous improvements and adaptations are crucial as we aim to further refine the model's accuracy and expand its applicability to larger and more diverse datasets.

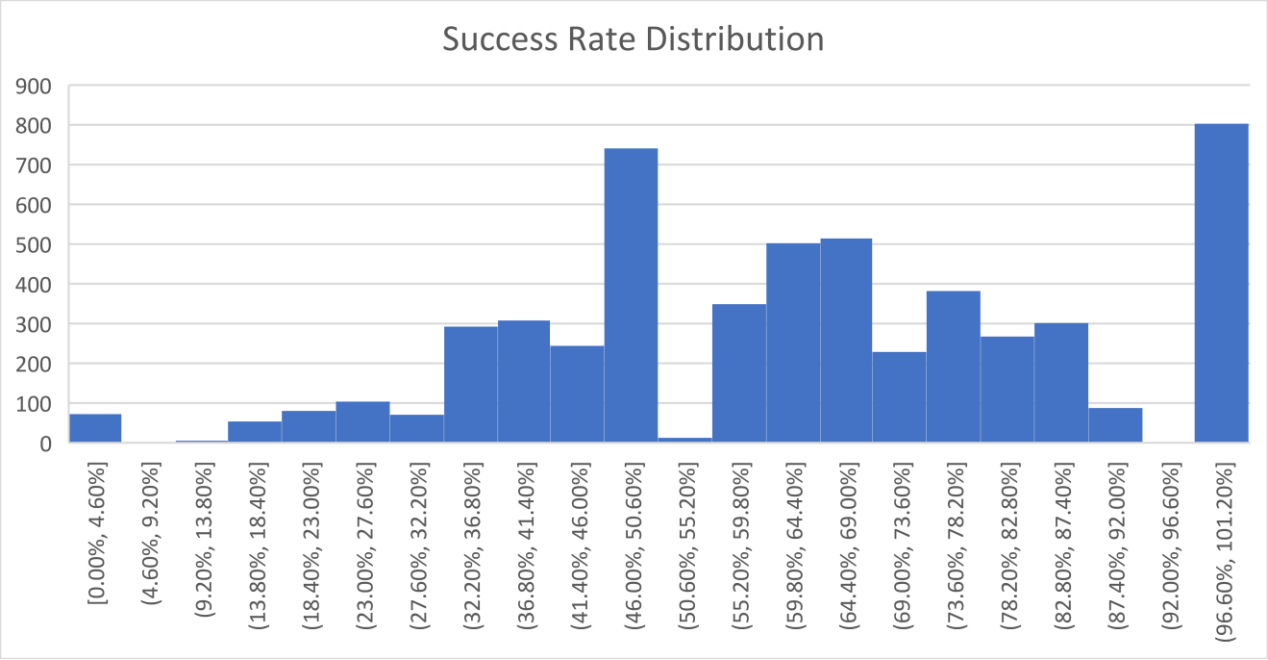


Figure 12. histogram of the distribution success rates.

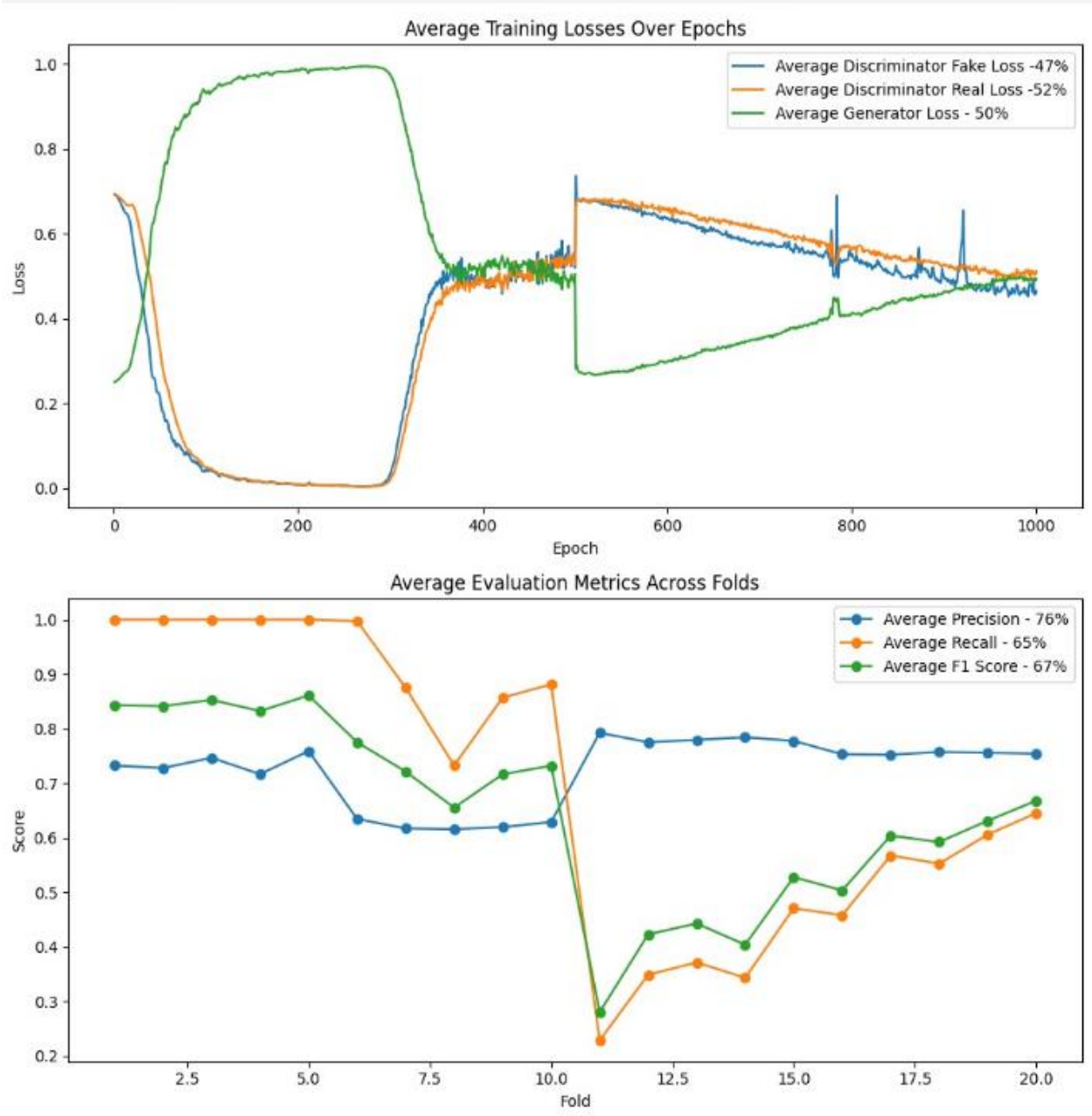


Figure 13. GAN Model Performance Analysis.

8. Learning Lessons: Reflecting on Project Execution and Future Changes

The development of our GAN-based citation prediction system provided numerous learning opportunities and insights into both the planning and execution stages of a complex machine learning project. Reflecting on the entire process, there are significant lessons we learned and aspects we would handle differently in retrospect.

8.1 Anticipating the Unavailability of Base Code

In Phase A, the theoretical part of our project, we operated under the assumption that we would have access to the implementation code of the article "A Link Prediction Algorithm Based on GAN," which our project is based on. It was only during Phase B, the implementation phase, that we realized no such code was available, forcing us to develop everything from scratch. This realization was not only a shock but also a major setback, consuming much more time and effort than anticipated. In retrospect, considering the potential unavailability of the code from the outset would have better prepared us for this scenario, potentially saving time and reducing stress.

8.2 Limited Flexibility for Different Datasets

Our solution was specifically designed and optimized for the Cora dataset, without considering the adaptability to other datasets. As the project progressed, it became evident that adapting our model to work with different datasets would require extensive modifications to the code, due to its lack of flexibility and generalization. Moving forward, designing the code to be more modular and dataset-agnostic from the beginning would enhance its utility and scalability, allowing for broader application without significant rework.

8.3 Challenges with the Hierarchical Network Layering Algorithm

The hierarchical network layering algorithm, NetLay, critical to our project, posed significant challenges. Described in detail in Section 6.1, the implementation adapted from the original article did not suit our specific requirements. We opted to use the Community Detection algorithm (described in Section 2.3) instead, but this adaptation only yielded three graph layers, with a drastic size reduction between the first and second layers. Efforts to moderate this size change were unsuccessful, and due to time constraints, we had to proceed with the existing structure. This limitation may have adversely impacted the model's performance and required a different approach. A more thorough initial testing and validation of the algorithm's applicability to our project could have allowed for alternative strategies to be developed earlier in the project timeline.

8.4 Learning Lessons Conclusion

These experiences underline the importance of flexibility in project planning and execution, especially in fields as dynamic and unpredictable as machine learning and AI. Each challenge also provided a valuable learning opportunity, highlighting areas for improvement in future projects, such as the need for early validation of assumptions and increased code modularity. These lessons will guide us in better anticipating project needs and adjusting methodologies to accommodate unexpected challenges in future work.

9. Evaluation of Project Benchmarks

Reflecting on the benchmarks set at the outset of our project, we believe that not only did we meet these expectations, but in many ways, we exceeded them. This achievement is particularly significant given the constraints and challenges we faced throughout the duration of the project.

9.1 Overcoming Initial Limitations

At the beginning of this project, our experience in Machine Learning was minimal, and our coding skills in Python were largely undeveloped. These factors initially set a modest expectation for what

we could achieve. However, despite these limitations, we managed to deliver a well-functioning project that addressed complex problems in citation prediction using advanced machine learning techniques. This accomplishment is a testament to our dedication and rapid skill acquisition under pressure.

Moreover, the unexpected realization that we would not have access to pre-existing code for the "A Link Prediction Algorithm Based on GAN" forced us to develop critical components from scratch. This challenge, detailed in Section 8.1, required us to quickly ascend a steep learning curve in both Python programming and machine learning concepts.

9.2 Managing Time Constraints and External Pressures

The academic calendar for this semester was impacted by the events of October 7th, resulting in a shortened semester of only 11 weeks instead of the usual 13. This reduction in time, coupled with our commitments to part-time jobs in the industry and other academic responsibilities, significantly constrained our available time for the project.

Despite these pressures, the outcome of the project exceeded our initial expectations. The successful completion under such tight constraints highlights our ability to efficiently manage time and prioritize tasks effectively. It also demonstrates our capacity to adapt to changing circumstances, a crucial skill in both academic and professional settings.

9.3 Evaluation Of The Project Benchmarks Conclusion

In conclusion, the benchmarks we set for ourselves at the beginning of this project were not only met but exceeded. This success was driven by our commitment to learning and adapting quickly, effective time management, and the ability to overcome significant initial handicaps in knowledge and experience. The project not only served as a valuable learning experience but also as proof of our capabilities to deliver complex solutions under challenging conditions.

10. References

[1]. Haiyan Jin, Guodong Xu *, Kangda Cheng, Jinlong Liu and Zhilu Wu ,
- A Link Prediction Algorithm Based on GAN , Harbin Institute of Technology, Harbin 150001 - MDPI electronics Journal. A Link Prediction Algorithm Based on GAN

[2]. Grover, A.; Leskovec, J. Node2vec: Scalable feature learning for networks. In Proceedings of the KDD' 16, San Francisco, CA,
USA, 13–17 August 2016; ACM Press: New York, NY, USA, 2016; pp. 1–10.
Node2Vec: Scalable Feature Learning for Networks

[3]. Kronmal, R.A.; Peterson, A.V., Jr. On the alias method for generating random variables from a discrete distribution. Am. Stat. **1979**, 33, 214–218.

[4]. Zhang, T.; Li, H.; Hong, W.; Yuan, X.; Wei, X. Deep first formal concept search. Sci. World J. **2014**, 2014, 275679.
Deep First Formal Concept Search , Deep first formal concept search

[5]. Wang, H.; Wang, J.; Wang, J.; Zhao, M.; Zhang, W.; Zhang, F.; Li, W.; Xie, X.; Guo, M. GraphGAN: Graph representation learning
with generative adversarial nets. IEEE Trans. Knowl. Data Eng. **2018**, 33, 3090–3103.
Learning Graph Representation With Generative Adversarial Nets

[6]. Lü, L.; Zhou, T. Link prediction in complex networks: A survey. Phys. A Stat. Mech. Its Appl. **2010**, 390, 1150–1170. Link Prediction in Complex Networks: A Survey

[7]. Medium. (n.d.). Synthetic data generation using generative adversarial networks (GANs) Part 2. Data Science at Microsoft. Retrieved [date you accessed the article], from GAN Architecture

[8]. Overview of deep learning on graph embeddings Towards data science

[9]. Rosvall, M., & Bergstrom, C. T. (2008). Maps of random walks on complex networks reveal community structure. arXiv preprint arXiv:0707.0609. Retrieved from Community Detection Algorithm