University of Southampton

Faculty of Engineering and Physical Sciences

Electronics and Computer Science

Exploring the use of Graph Neural Networks for Link Prediction

by

Jozef Cibicek

23 October 2020

Supervisor: Dr. Markus Brede
Second Examiner: Prof. Eric Rogers

A dissertation submitted in partial fulfilment of the degree
of MSc in Computer Science

University of Southampton

ABSTRACT

FACULTY OF ENGINEERING AND PHYSICAL SCIENCES
ELECTRONICS AND COMPUTER SCIENCE

Master of Science

by Jozef Cibicek

The recent breakthroughs in deep learning from the early 2010's have greatly popularised machine learning on euclidean structures such as images, text or pure numerical data. More recent deep learning models, of Geometric deep learning, Bronstein et al. (2017) are applied in supervised and unsupervised manner to graphs, manifolds or other uneven structures with noticeable success.

Link Prediction in complex networks, first introduced by Liben-Nowell and Kleinberg (2004) remains a widely studied, interdisciplinary topic tackled by approaches of various knowledge domains. Whereas a vast number of solutions is applied, issues such as scalability, time complexity or resource heavy usage remain present and complicate the shift to ever increasing throughput of data. Graph Neural Networks, a subset of geometric deep learning is a connectionist model able to perform Link Prediction on such complex networks.

This work brings a thorough analysis of state of the art GNN models (GCN, Graph-SAGE, GAT, GIN, PGNN and SEAL) including training and testing the models on 10 undirected, static graph datasets from varying domains twice, first without any explicit node features, and second time, with explicit node features, namely; Graphlet Degree Signatures proposed by Milenković and Pržulj (2008). A total number of 105,000 iterations of training and testing is visualised and documented with appropriate learning metrics.

The five featureless Graph Neural Network models score on average 0.819 Test AUC and 0.507 Test loss on the collection of 10 undirected static graph datasets. The addition of Graphlet Degree Signatures as explicit node features improves the Test AUC by 3 points to 0.851 Test AUC and visibly improves the learning process.

Graph Neural Networks applied to Link Prediction attain almost perfect results in a time efficient manner, and combined with the extracted orbital features they outperform traditional state of the art Link Prediction approaches in terms of model accuracy and scalability to larger graphs.

Acknowledgements

First and foremost, I would like to thank my parents that supported me throughout my studies.

Next, I would like to thank my supervisor, Dr. Markus Brede who allowed me to devise and pursue this topic and guided me through its completion.

Then, I would like to thank the founders of Vast.AI whose brilliant idea of sharing peer to peer cloud GPUs have made this thesis economically possible.

Last but not least, I would like to thank the developers of Facebook AI Research and SNAP group of Stanford University for releasing wonderful implementations of the actual GNN models used in this work.

I hope the results of this work inspire others to explore the power and the creative potential of Graph Neural Networks and Geometric Deep Learning in general.

Use of Software and Data

I have used the official implementations of all GNN models in this work, namely: SEAL with DGCN, GCN, GraphSAGE, GAT, GIN, P-GNN. SEAL is attributed to Zhang and Chen (2018) and the remaining models to the SNAP Group of the Stanford University Leskovec and Sosič (2016). These models were trained to produce all of the figures with AUC and loss curves.

I have also used the compiled binaries for the orbital feature computation, which is attributed to the github repository of Hocevar (2016).

The general backbone of this work, such as the models themselves, dataset building, or training of the models is built on an extension of Pytorch Paszke et al. (2017), called Pytorch Geometric Fey and Lenssen (2019).

Except this, the following software libraries were used throughout the work for various auxiliary tasks such as results plotting Matplotlib Hunter (2007), general data manipulation Pandas pandas development team (2020), graph manipulation Networkx Hagberg et al. (2008) and array programming Numpy Harris et al. (2020).

The datasets and their respective authors are referenced in the Table 3.1 under the Methodology section.

The actual work itself was written on Mac OS implementation of TexShop in LaTeX. The code was ran in Jupyter Notebook environment on Vast.AI cloud machines.

Statement of Originality

- I have read and understood the ECS Academic Integrity information and the University's Academic Integrity Guidance for Students.
- I am aware that failure to act in accordance with the Regulations Governing Academic Integrity may lead to the imposition of penalties which, for the most serious cases, may include termination of programme.
- I consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to verify whether my work contains plagiarised material, and for quality assurance purposes.

You must change the statements in the boxes if you do not agree with them.

We expect you to acknowledge all sources of information (e.g. ideas, algorithms, data) using citations. You must also put quotation marks around any sections of text that you have copied without paraphrasing. If any figures or tables have been taken or modified from another source, you must explain this in the caption and cite the original source.

I have acknowledged all sources, and identified any content taken from elsewhere.

If you have used any code (e.g. open-source code), reference designs, or similar resources that have been produced by anyone else, you must list them in the box below. In the report, you must explain what was used and how it relates to the work you have done.

I have used open source software and data in my work and I have acknowledged and explained it in the section: Use of Software and Data

You can consult with module teaching staff/demonstrators, but you should not show anyone else your work (this includes uploading your work to publicly-accessible repositories e.g. Github, unless expressly permitted by the module leader), or help them to do theirs. For individual assignments, we expect you to work on your own. For group assignments, we expect that you work only with your allocated group. You must get permission in writing from the module teaching staff before you seek outside assistance, e.g. a proofreading service, and declare it here.

I did all the work myself, or with my allocated group, and have not helped anyone else.

We expect that you have not fabricated, modified or distorted any data, evidence, references, experimental results, or other material used or presented in the report. You must clearly describe your experiments and how the results were obtained, and include all data, source code and/or designs (either in the report, or submitted as a separate file) so that your results could be reproduced.

The material in the report is genuine, and I have included all my data/code/designs.

We expect that you have not previously submitted any part of this work for another assessment. You must get permission in writing from the module teaching staff before re-using any of your previously submitted work for this assessment.

I have not submitted any part of this work for another assessment.

If your work involved research/studies (including surveys) on human participants, their cells or data, or on animals, you must have been granted ethical approval before the work was carried out, and any experiments must have followed these requirements. You must give details of this in the report, and list the ethical approval reference number(s) in the box below.

My work did not involve human participants, their cells or data, or animals.

ECS Statement of Originality Template, updated August 2018, Alex Weddell aiofficer@ecs.soton.ac.uk

Contents

List of Figures

List of Tables

Acronyms

LP - Link Prediction

GNN - Graph Neural Network

RecGNN - Recurrent Graph Neural Network

GCN - Graph Convolutional Network

N2V - Node2Vec

GAE - Graph AutoEncoder

EML - Euclidean Machine Learning

GML - Geometric Machine Learning

VGAE - Variational Graph AutoEncoder

NE - Network Embedding

SAGE - GraphSAGE Neural Network

DGCN - Deep Graph Convolutional Network

GAT - Graph Attention Network

GIN - Graph Isomorphism Network

PGNN - Position Aware Graph Neural Network

Chapter 1

Introduction

1.1 Background and Motivation

Link prediction is now an established problem of network analysis which aims to infer whether connections are formed between the network's members. Originally formulated in social network analysis in Liben-Nowell and Kleinberg (2004), Link Prediction has been inter-disciplinarily studied and applied across various problem domains including biomedical sciences, recommendation algorithms or IoT networks.

Traditionally performed using heuristic methods such as the Common Neighbours, Katz and Jaccard indices, or Pagerank and its variations, the mentioned heuristic methods operate under specific assumptions about the behavior of the network, which might not be transferable to other network types. A more generalisable approach is to attempt automatic inference of the network connections given the recent rise of data volume and success of supervised machine learning models.

Last decade witnessed a tremendous rise in application of traditional neural networks. Classical neural network architectures have been successfully applied to problem domains of numerical, visual, textual, and auditory natures. The processed datum was naturally represented in euclidean feature spaces, which was embraced as a standardised procedure. In recent years, the idea of learning machines operating on euclidean features was extended further, allowing to capture structures of richer geometrical shapes such as graphs Bronstein et al. (2017). This advancement gave birth to a 'connectionist' model processing graphs called Graph Neural Network.

Being a relatively new field, Graph Neural Networks were first proposed by Scarselli et al. (2008). Its' sheer connectionist structure of a graph like data makes it reasonable to apply Graph Neural Networks to the problem of Link Prediction.

The highly interconnected nature of the modern world, combined with the undisputed success of supervised learning models, justifies the exploration of Link Prediction performed by Graph Neural Networks.

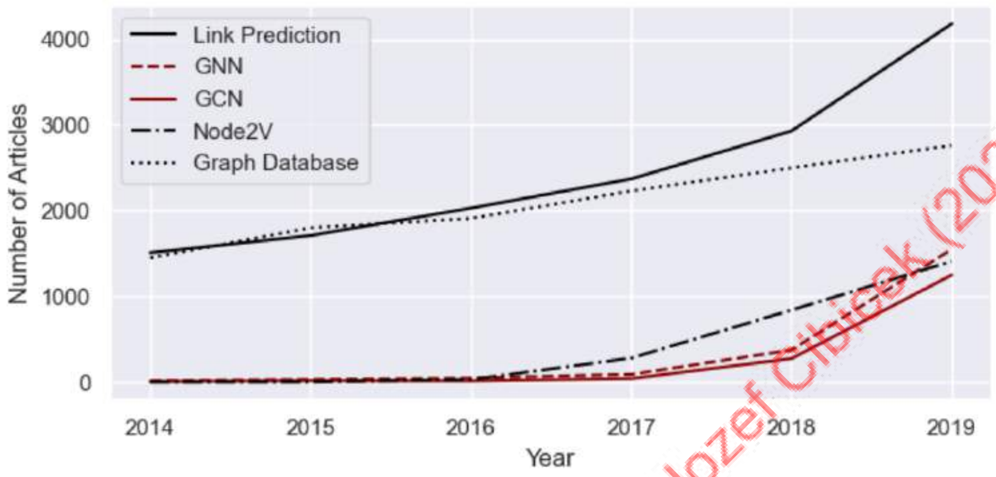


FIGURE 1.1: Historic keyword occurrences in articles published to Google Scholar between 2014 and 2019. Extracted with 'Academic Keyword Occurrence' Strobel (2018)

1.2 Aims and Contributions

This work predominantly aims to explore the usage and applicability of Graph Neural Networks for Link Prediction in the case of undirected static networks. More specifically, it focuses on the performance of Graph Neural Networks without and with explicit node features. Initially, the work beings with application of 4 heuristics as a baseline approach to Link Prediction. Then a collection of five state of the art GNN models are trained and tested on 10 datasets. Eventually, node features (Graphlet Degree Signature Vectors) are generated and are supplied to the same GNN models.

The following questions are answered in this work:

- To which extent are GNNs applicable for Link Prediction in undirected, static and homogeneous graphs?
- How does the GNN itself conducts the prediction of a link?
- Are GNN's more effective at Link Prediction against traditional methods?
- How well a GNN learns to predict a link without supplied explicit node features?

Prediction accuracy of both GNN and traditional methods, as well as their behaviour during the learning procedure is measured and compared.

As the Graph Neural Networks are known to include an extensible number of architectures and latent embedding methods, the work explores the context of various state of the art Graph Neural Network architectures and feature extraction methods and compares their performance.

- Which are the most effective GNN architectures for Link Prediction in graphs?

GNN architectures such as Graph Convolutional Networks, Graph Attention Networks Gu et al. (2019). Latent embedding methods such as DeepWalk or Node2Vec are omitted, as they are Network Embeddings and not being fully developed GNN models.

The final part of the work extends the explored GNN models and tries to answer the question:

- How the explored GNN models can be extended to further increase performance in Link Prediction?

The work implements and evaluates feature extraction method based on graphlet counts and graphlet correlation distances, as proposed in Milenković and Pržulj (2008).

1.3 Scope and Focus

The world of graph theory, combined with link prediction and graph neural networks covers undoubtedly wide research space. To reduce the complex scope of the project, it is important to set some boundaries beforehand and specify the main areas of the project's focus. In general, a common pattern found throughout this work is to utilise a notion of popularity in the selection of the matters, selecting the most known datasets, techniques or architectures.

1.3.1 Subject Matter: Graph Data

Graph data comes in various forms and types. In its simplest case, a graph is a collection of nodes and edges. Nodes, as well as edges can have properties which dramatically increase the graph's complexity. This work focuses on undirected, static graphs, without any temporal properties.

In the context of this analysis, a graph can be also thought of a dataset, which is the subject of the experiments present in this work. To impose generalisation, the selected datasets are from different problem domains, such as ecology, financial markets or protein to protein interactions in order not to be limited to a specific network behaviour. The work reviews popular datasets such as USAir, EColi, Yeast, or Arxiv. The featureless and static attributes of these datasets follow a more real world use-case where its very probable to encounter these of graphs rather than directed, temporal or with features.

1.3.2 Problem to Analyse: Link Prediction

Link Prediction is a problem domain linked to extensive number of disciplines, ranging from Physics through Medical Sciences to Economics. The number of backgrounds also brings a number of disjoint solutions. To this day, there are around 50 different heuristic methods published, each with varying performances on different datasets Kumar et al. (2020) This work reviews and focuses on the most popular heuristics used for Link Prediction, such as the following popular heuristic methods, namely; Jaccard Index Jaccard (1901), Preferential Attachment Zhou et al. (2009), Adamic-Adair index Adamic and Adar (2003), and Resource Allocation Index Newman (2001).

1.3.3 Explored Approach: Graph Neural Networks

Graph Neural Networks became a fast evolving field with numerous architectures published frequently. The type of the graph data above determines the complexity of the model's architecture, and the notion of popularity selects the most popular GNN architectures, such as Graph Convolutional Networks Kipf and Welling (2016a), Deep Graph Convolutional Networks Zhang et al. (2018), or Graph Attention Networks Veličković et al. (2017).

The explored models are: GCN Kipf and Welling (2016a), GraphSAGE Hamilton et al. (2017), GAT Veličković et al. (2017), GIN Xu et al. (2018), SEAL Zhang and Chen (2018), PGNN You et al. (2019).

Chapter 2

Related Work

This chapter introduces and describes the problem of link prediction, followed by comparing the most popular solutions used in link prediction research. Later on, it proceeds by introducing the topic of Graph Neural Networks and summarises the state of the art GNN architectural types. The chapter finishes with a summary about the application of Graph Neural Networks in link prediction and describes the state of the art GNN models applicable for LP.

2.1 Link Prediction - Introduction

Before its seminal formulation in Liben-Nowell and Kleinberg (2004), the problem of predicting relationships in graph structures has existed in various forms for some time, mainly in the field of network analysis. Predating works Kleinberg (1998), Barabási and Albert (1999), Newman (2001) or Adamic and Adar (2003) explore the problem of predicting links from evolutionary network perspective, however lacking an unified framework for performance evaluation Liben-Nowell and Kleinberg (2004).

The seminal work on Link Prediction Liben-Nowell and Kleinberg (2004) formulates the Link Prediction problem as to accurately predict the edges that will be added to a network N during the interval from time T_1 to a given future time T_2. In future works, this original formulation is amended, and Link Prediction can be also viewed as a network reconstruction problem, where the goal is to infer missing links from the network rather than predicting new ones. Rather than focusing on the future connections between the members of the network, Link Prediction has been also used for an identification of spurious links, or for network reconstruction.

2.1.1 Evaluation Metrics

Link prediction performed on graph data is commonly evaluated by either Area under Curve (AUC) or Average Precision (AP) metrics. In order for the metrics to be computed, the set of all known links is divided into Training Set E^T and Validation Set E^P which is not used in the training procedure. Generally, the AUC metric measures the classification ratio of true positives to false positives, or in other words, the number of correctly classified examples to incorrect ones. As there are no false positives in link prediction, the false positive rate is denoted as the number of non-existing links that were recognised by the classifier as existing ones. More specifically, the AUC can be computed as:

$$AUC = \frac{n' + 0.5n''}{n} \tag{2.1}$$

where the n represents the number of independent comparisons, n' the number of times a validation link has higher score than a non-existent link, n" number of times these two scores are equal Lü and Zhou (2011). An AUC below or equal to 0.5 can be thought of as random guess. Average Precision is the ratio of right links over all predicted ones Lü and Zhou (2011).

$$AP = \frac{Lr}{L} = \frac{|\{links|score > 0.5, \in E^T\}|}{|\{links|score > 0.5\}|} \tag{2.2}$$

To further increase generalisation in estimation, the experiments are performed using K-fold, typically 10-fold cross-validation.

2.2 Methods for Link Prediction

Since its inception, the problem of Link Prediction has been studied in multiple disciplines (Physics, Sociology, Computer Science, Mathematics), resulting in a number of different approaches. While the solution set contains approaches of varying complexity, the simplest and empirically most effective approaches tend to be the heuristic ones. Although efficient, these methods are numerous and suffer from their inability to generalise well across different problem domains and datasets.

In 2010, a comprehensive survey on Link Prediction published by Zhou and Lu Lü and Zhou (2011) divides the Link Prediction approaches into the following three categories (1-3), however, list was extended in the next 10 years by more state of the art approaches such as Network Embeddings or Graph Neural Networks (4-5) Kumar et al. (2020).

2.2.1 Similarity-Based Algorithms

The most simple approaches for link prediction operate within the vicinity of the nodes of interest, utilising an assumption that a given pair of target nodes has a stronger probability of linkage if the nodes have a joint set of same neighbouring nodes. Typical example is a **Common Neighbours** heuristic that counts the cardinality of a set consisting of an intersection between the node's neighbourhood sets. Adamic Adar Index can be seen as a weighted modification to the Common neighbour heuristic, punishing more connected neighbours by assigning more weight to the less connected ones. Global Similarity Indices process the entire network, by taking into account ensembles of all possible edges and nodes. Katz index computes the Katz (1953) SimRank is a modification of PageRankJeh and Widom (2002)

2.2.2 Maximum Likelihood Methods

As the name suggests, ML methods compute the likelihood of any non-observed link from the maximum likelihood of the network structure. Popular examples are Stochastic Block Model or the Hierarchical Structure model. Maximum likelihood methods excel at analysis of small networks, where they correctly identify the areas of interest, however, their performance on larger graphs is much limited due to their enormous time complexity.

2.2.3 Probabilistic Models

The most popular probabilistic methods for Link Prediction are built on bayesian inference or combinatorial optimisation. The applicability of probabilistic models is hindered by their low scalability to large networks, as they require a time consuming calculation of structural, as well as node or edge information Kumar et al. (2020).

2.2.4 Dimensionality Reduction Methods

A collection of approaches to Link Prediction, based on various forms of matrix factorisation. Link prediction can be also formulated as matrix completion problem, solved either non-negative matrix factorisation NNMF or Singular Value Decomposition SVD. After solving the optimisation problem, the similarity matrix storing the probability of individual links can be computed Menon and Elkan (2011).

Network Embeddings Grover and Leskovec (2016) Perozzi et al. (2014) Tang et al. (2015) compute the latent representation of nodes, which can be subsequently used for link prediction or classification tasks. Network Embeddings essentially emulate the logic of word embeddings in graph data and are well suited for some Link Prediction scenarios despite their transductive properties.

2.2.5 Learning Algorithms

More recent approaches for Link Prediction utilise empirically successful learning paradigms. Learning algorithms typically follow the optimisation of an objective function, which is minimised in order to find a particular configuration of algorithm weights that lead to the lowest loss. Such methods can be divided into Euclidean Machine Learning, which operates on standard, grid-like feature space, or novel Geometrical Machine Learning methods, which extract features directly from graph structures, manifolds or point clouds.

Euclidean machine learning approaches are usually well known classification algorithms, such as SVM Al Hasan et al. (2006), Decision Trees or a Multi Layer Perceptron. Regarding approaches that are related to the problem domain, the Weifseiler-Lehman machine for Link Prediction, Zhang and Chen (2017) is inspired by the 1968 Weisfeiler-Lehman graph isomorphism test Boris and Lehman (1968).

Graph Neural Networks, an example of GML, typically perform graph convolution on the node neighbourhoods, extracting the features which are later aggregated under the message passing framework. The GNNs are the main area explored in this work; GCN Kipf and Welling (2016a), GraphSage Hamilton et al. (2017), GAT Veličković et al. (2017), GIN Xu et al. (2018), DGCN and SEAL Zhang et al. (2018) Zhang and Chen (2018), PGNN You et al. (2019).

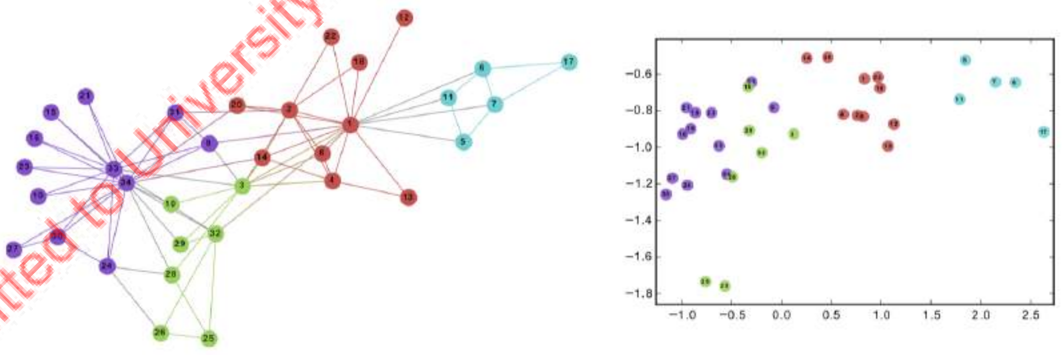


FIGURE 2.1: Zachary's Karate club Zachary (1977) network represented in latent space by DeepWalk Network Embedding. Perozzi et al. (2014)

2.3 Graph Neural Networks

Graph Neural Networks are a semi supervised learning model belonging to the umbrella term of Geometrical Learning. The origins of Graph Neural Networks can be traced to the works of Gori et al. (2005) that coined the term Graph Neural Networks in 2005, and formalised the first learning paradigm in 2008 Scarselli et al. (2008).

2.3.1 Neighbourhood Aggregation

The modus operandi of most modern GNN models can be described by a neighbour aggregation (or message passing) scheme where the feature representation of a given node is an aggregation of the feature vectors of the node's neighbouring nodes.

More formally, the updated node's state $h_t{}^n$ can be represented as:

$$h_t^n = q\left(h_{t-1}^n, \bigcup_{nj:n->nj} f_t(h_{t-1}^n, k, h_{t-1}^{n_j})\right)$$

where, the union symbol denotes aggregation in this case and can be replaced by various aggregation types such as plain Kipf and Welling (2016a) or LSTM-like Hamilton et al. (2017).

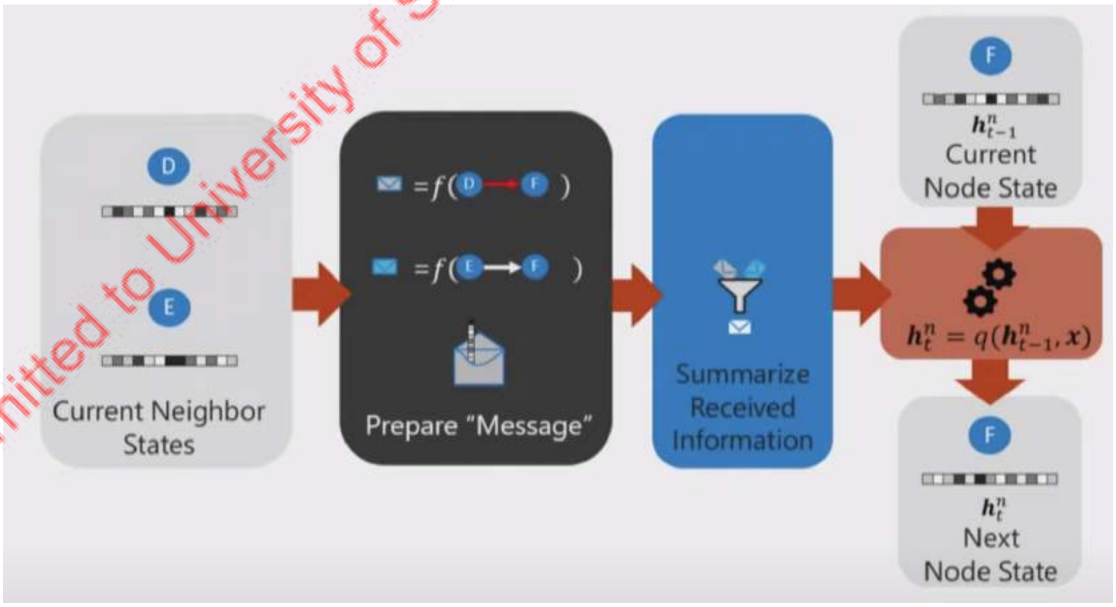


FIGURE 2.2: Flowchart of Message Passing / Neighbour Aggregation algorithm. mic

Over the years, the field of Graph Neural Networks departed from its seminal, recurrent architecture and saw innovations in approaches and architectural types. As of late

2019, Wu et al. (2020) summarise the developed GNN architectures in the following 4 categories.

1. Recurrent Graph Neural Networks
2. Convolutional Graph Neural Networks
   - (a) Spatial methods
   - (b) Spectral methods
3. Graph AutoEncoders
   - (a) Network Embeddings
   - (b) Graph Generation Approaches
4. Spatial-temporal Graph Neural Networks

2.3.2 Recurrent Graph Neural Networks

Recurrent graph neural networks or RecGNNs is the original class of approach published by Scarselli et al. (2008). RecGNNs iterates over graph nodes, in a similar manner as the RNN models of Euclidean Machine Learning. In the recent years, RecGNNs left the academic spotlight as they were later replaced by Convolutional GNN's due to their high computational cost.

2.3.3 Convolutional Graph Neural Networks

Originally proposed by Kipf and Welling (2016a), Convolutional Graph Neural networks are became the most established variant of the modern GNNs. As the name suggests, ConvGNN bear similarities with Convolutional neural networks of Euclidean Machine Learning by performing automatic feature extraction. Depending on the type of convolution, the ConvGNNs are divided into Spectral and Spatial categories.

Spatial convolutional methods, similarly to the traditional convolutional neural networks perform feature extraction by a fixed window on the input data, however, this time the window captures K wide neighbourhood information of each individual graph node.

More complex Spectral ConvGNNs compute an eigen-decomposition of the Laplacian matrix of the constituent graph. Based upon the spectral graph theory, the decomposition unfolds important, behavioural properties about the network, which are subject to an analysis in the Fourier space. Popular examples are ChebNet published in 2016 by Defferrard et al. (2016) or Bruna et al. (2013).

2.3.4 Graph AutoEncoders

Graph AutoEncoders rely on an analogy of traditional AutoEncoders Kramer (1991) where an Encoding and Decoding part learn to copy its input to its output. GAE's aim to achieve the most error-less reconstruction of a graph by the means of latent representation and can be further divided into Network Embedding or Graph Generation approaches.

Network Embedding GAE's (NEGAE) learns network embedding by using an encoder to learn network embeddings and an decoder to reconstruct the latent embeddings into its original graph structure. They must not be confused with The Network Embeddings below, which are designed mainly for learning Network Embeddings. The encoders of NEGAE's consist of convolutional layers, though the decoders computes the pairwise distance given network embeddings Wu et al. (2020). Speaking of training process, the task is to minimise the distance between the original adjacency matrix and the reconstructed one Wu et al. (2020).

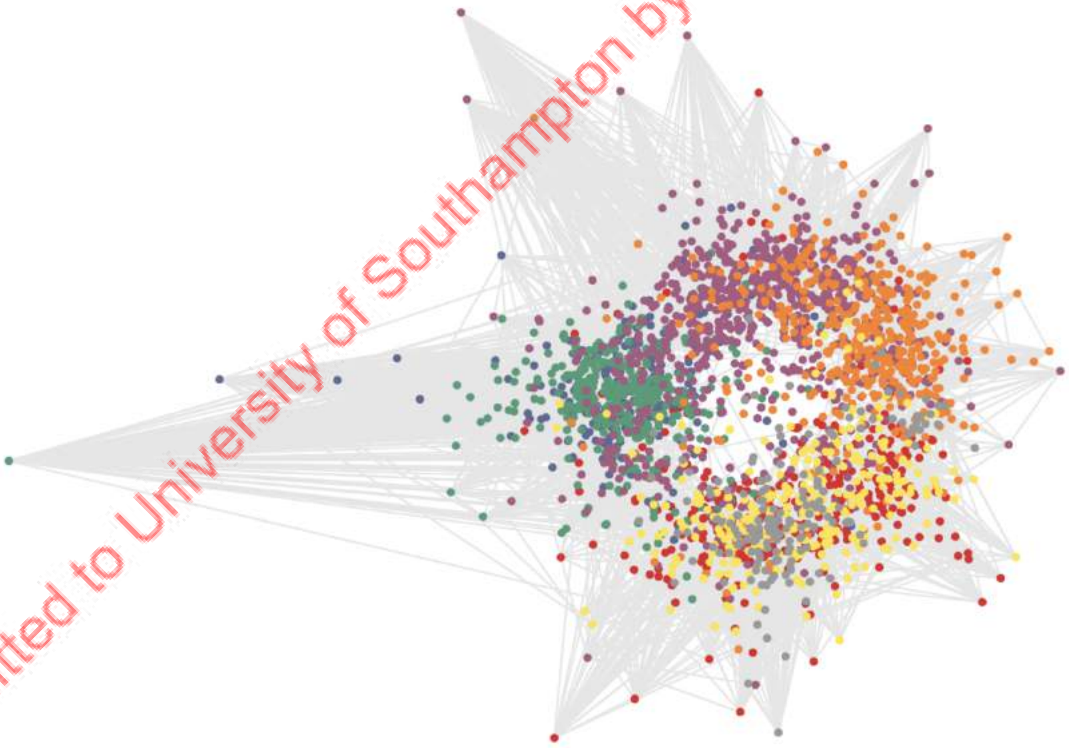


FIGURE 2.3: Latent space of VGAE trained on Cora dataset Kipf and Welling (2016b)

Graph Generation (GAE) approaches learn the generative distribution of the graphs (similarly to VAE) and generate graphs from this distribution. Sequential GG GAE's output the graph step by step while Global GG GAE's generate a graph at one timestep.

2.3.5 Spatial-temporal Graph Neural Network

A more recent GNN architecture, Spatial Temporal Graph Neural Networks operate on Spatial Temporal Graphs, where the a graph convolution captures the network at different timesteps. This architecture type is not further explored in this work as the work does not explore temporal graph data.

2.4 Graph Neural Networks & Link Prediction

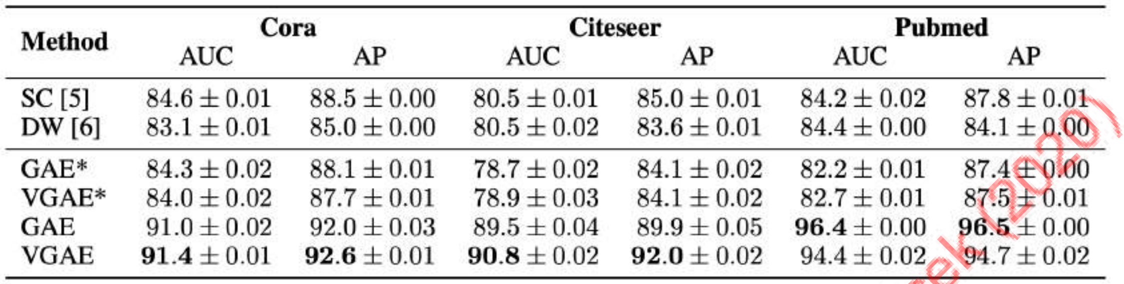
2.4.1 State of the Art

Although the Graph Neural Networks were first proposed in 2005 Gori et al. (2005) and finalised in 2008 by Scarselli et al. (2008), the original works have not focused on their applications for Link Prediction. At the same time, a number of works started applying traditional, euclidean machine learning to graph related tasks achieving noticeable success. However, it is more intuitive to leverage the learning procedures to their full advantage by restructuring them to the non-euclidean, geometrical features of a graph.

The beginnings of GNN research for LP can be traced to first half of 2010's, where the success of word embeddings Mikolov et al. (2013) generalised into network embeddings brought DeepWalk Perozzi et al. (2014) LINE Tang et al. (2015) Node2Vec Grover and Leskovec (2016) with results outperforming traditional heuristic methods. Although well performing, these novel approaches based on matrix factorisation perform transductive learning, which needs to retrain the model if the data modifies. As the graph data is naturally dynamic and changing, this presents an undesirable aspect to transductive learning paradigms, by leading to zero reusability and the inability to generalise to unseen nodes.

Although both approaches are used for similar graph related tasks, the main distinction between GNN's and Network Embeddings is that a GNN is full purpose, end to end machine learning model while Network Embedding is a specific method aimed to represent a graph in its most optimal latent representation.More specifically, while both methods are used to perform graph classification, clustering or recommendation tasks, Network Embeddings need to utilise additional machine learning algorithms such as SVM's or MLP, while GNN's do this automatically. Nevertheless, these Network Embeddings were detrimental in the following academic work on Graph Learning, being used as a basis for the architectures to come.

The first notable publication directly incorporating GNN's rather than NE's for Link Prediction saw light in 2016, the VGAE Kipf and Welling (2016b) incorporates node features into its learning paradigm. VGAE is based on an Convolutional layered Encoder and a simple, inner product based decoder which reconstructs the latent representation into graph input data. As the Node2Vec Grover and Leskovec (2016) was not known at that time, the VGAE's comparison on LP tasks includes only DeepWalk Perozzi et al. (2014) and Spectral Clustering benchmarks Tang and Liu (2011). VGAE and GAE beat SC and DeepWalk Perozzi et al. (2014) on all datasets with both AUC and AP metrics.



Method	**Cora**		**Citeseer**		**Pubmed**	
	AUC	AP	AUC	AP	AUC	AP
SC [5]	84.6 ± 0.01	88.5 ± 0.00	80.5 ± 0.01	85.0 ± 0.01	84.2 ± 0.02	87.8 ± 0.01
DW [6]	83.1 ± 0.01	85.0 ± 0.00	80.5 ± 0.02	83.6 ± 0.01	84.4 ± 0.00	84.1 ± 0.00
GAE*	84.3 ± 0.02	88.1 ± 0.01	78.7 ± 0.02	84.1 ± 0.02	82.2 ± 0.01	87.4 ± 0.00
VGAE*	84.0 ± 0.02	87.7 ± 0.01	78.9 ± 0.03	84.1 ± 0.02	82.7 ± 0.01	87.5 ± 0.01
GAE	91.0 ± 0.02	92.0 ± 0.03	89.5 ± 0.04	89.9 ± 0.05	$\mathbf{96.4} \pm 0.00$	$\mathbf{96.5} \pm 0.00$
VGAE	$\mathbf{91.4} \pm 0.01$	$\mathbf{92.6} \pm 0.01$	$\mathbf{90.8} \pm 0.02$	$\mathbf{92.0} \pm 0.02$	94.4 ± 0.02	94.7 ± 0.02

FIGURE 2.4: Results of Kipf's VGAE on LP Kipf and Welling (2016b). The results display the state of the art GNN performance for Link Prediction in 2016.

On the contrary to the previous transductive paradigms, Hamilton et al. (2017) propose GraphSage, an inductive representation learning paradigm, which captures the positional information whenever a graph is modified and generalises to unseen nodes. Structurally speaking, GraphSage is built a top of GCN, but learns more complex aggregator functions than simple convolutions ie. (LSTM...) Hamilton et al. (2017)

Graph Attention Networks Veličković et al. (2017) challenge transductive, as well as inductive problems by leveraging node attention attributes Bahdanau et al. (2014). GAT's are comprised solely of Graph Attention layers, which compute the hidden representations of nodes by attending its neighbours, following self-attention strategy Veličković et al. (2017). However, GATs, as well as GraphSage are unable to distinguish between symmetric nodes of the network, generating identical node embeddings for a pair of nodes which have the same local neighbourhood structure but different position in the graph You et al. (2019).

Zhang and Chen (2018) apply a modification of DGCN Zhang et al. (2018) for link prediction by learning features from local enclosing subgraphs instead of entire networks. Unlike other works, SEAL is a framework strictly for link prediction, and unifies a wide range of heuristic methods which are passed to GNN in addition with its adjacency matrix A. The main disadvantage of SEAL is the computation of the node features, which are transductive in nature, and need to be recomputed every time the graph has changed.

Xu et al. (2018) generalise the Weisfeiler-Lehman test 1968 Boris and Lehman (1968) for graph isomorphism to prove the maximally powerful discrimnative GNN, and formalise this proof into a novel architecture; Graph Isomorphism Networks. Satisfying the conditions of the WL test, the GIN is claimed to be the best performing GNN built on the

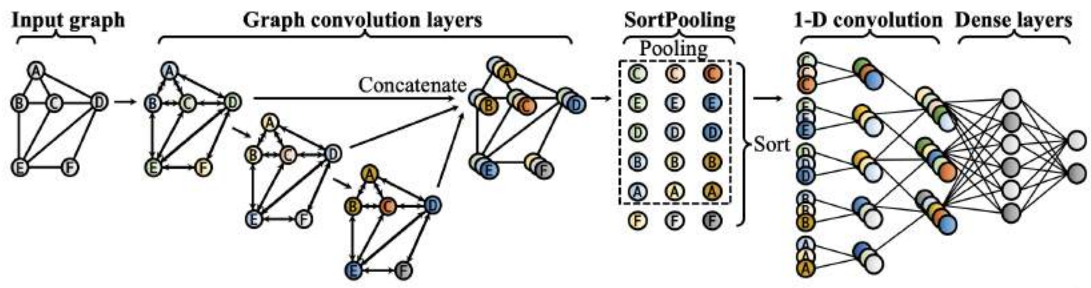
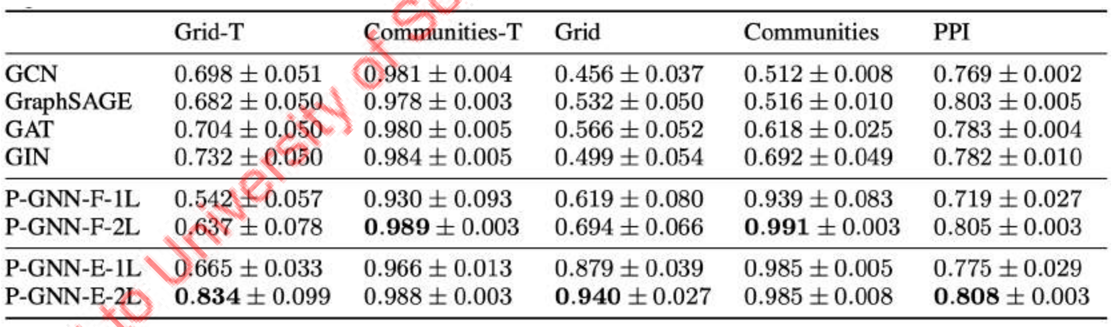


Figure 2.5: DCGNN architecture, as illustrated in Zhang and Chen (2018). The model head consists of a two layer MLP, having 50% dropout probability and RELU activations.

message passing framework Xu et al. (2018). GIN aggregates over multisets, which are sets of repeating elements representing the node neighbourhoods.

Position Aware Graph Neural Networks (PGNN) by You et al. (2019) capture individual node information with respect to its entire graph, allowing to perform inductive learning scalable to larger graphs. P-GNN achieves this by sampling a random subset of nodes (called anchor nodes), rather than capturing information from the local neighbourhoods. Then, the message aggregation is performed across all nodes in order to find ones with different positions in the network. PGNN scores higher than GCN Kipf and Welling (2016a), GraphSage Hamilton et al. (2017), GAT Veličković et al. (2017) and GIN Xu et al. (2018) in all four datasets for Link Prediction,



	Grid-T	Communities-T	Grid	Communities	PPI
GCN	0.698 ± 0.051	0.981 ± 0.004	0.456 ± 0.037	0.512 ± 0.008	0.769 ± 0.002
GraphSAGE	0.682 ± 0.050	0.978 ± 0.003	0.532 ± 0.050	0.516 ± 0.010	0.803 ± 0.005
GAT	0.704 ± 0.050	0.980 ± 0.005	0.566 ± 0.052	0.618 ± 0.025	0.783 ± 0.004
GIN	0.732 ± 0.050	0.984 ± 0.005	0.499 ± 0.054	0.692 ± 0.049	0.782 ± 0.010
P-GNN-F-1L	0.542 ± 0.057	0.930 ± 0.093	0.619 ± 0.080	0.939 ± 0.083	0.719 ± 0.027
P-GNN-F-2L	0.637 ± 0.078	$\mathbf{0.989} \pm 0.003$	0.694 ± 0.066	$\mathbf{0.991} \pm 0.003$	0.805 ± 0.003
P-GNN-E-1L	0.665 ± 0.033	0.966 ± 0.013	0.879 ± 0.039	0.985 ± 0.005	0.775 ± 0.029
P-GNN-E-2L	$\mathbf{0.834} \pm 0.099$	0.988 ± 0.003	$\mathbf{0.940} \pm 0.027$	0.985 ± 0.008	$\mathbf{0.808} \pm 0.003$

Figure 2.6: Results of You et al. (2019) Link Prediction task. T denotes the transductive version of the dataset. The GCN architecture is a replica of VGAE from Kipf and Welling (2016b)

2.4.2 Summary of the Literature Review

Over the last four years, the GNN's have greatly advanced their capabilities for Link Prediction. The field started with applying transductive Network Embeddings which required the model to be retrained (each time the graph has been modified) and combined with additional algorithms to perform predictions. The embeddings were later replaced by pure end to end GNN architectures built for node classification, initially transductive Kipf and Welling (2016b) Kipf and Welling (2016a), Hamilton et al. (2017), but later inductive Xu et al. (2018) You et al. (2019).

Eventually, GIN Xu et al. (2018) embody the theoretical proof of maximally powerful GNN's utilising the message passing framework while the recent PGNN You et al. (2019) allows both inductive and transductive learning processing, as well as retaining the positional information of nodes.

Chapter 3

Methodology

This chapter includes the conducted experiments for the work. It begins with a description of the process, followed by the description of the datasets and the parameters of the models. The experiment starts with a baseline comparison of 4 popular heuristics, which are then compared to a collection of state of the art GNNs in link prediction on 10 graph datasets. The results of the models are eventually summarised, towards the end of the section. The section ends with an attempt to improve the model performance using graphlet node degree features, as proposed in Milenković and Pržulj (2008).

3.1 Process Description - Link Prediction

This section describes the process of link prediction classification from dataset loading to evaluating model performance.

1. An arbitrary undirected graph dataset is loaded in edge pair format. A common way is to store datasets in .txt files, in two column format (In degree node - out degree node).

2. The original graph dataset is split into a training and testing split, where both splits have the same number of nodes as the original dataset, but less number of links. The removed links are reserved as positive links for the training process, representing the ground truth. Both splits sample a percentage of positive and negative (false) links of the original Graph.

   In some libraries, the link information is stored in a collection of binary masks, denoting the presence or absence of a link in a given split.

   The point of the algorithm is to learn the underlying structure of the graph, without the full edge information.

3. Train and Test split contains positive and negative samples.

4. The GNN models are usually used for node classification, therefore the architecture has to be amended for link prediction. More specifically, the model's 'head' is altered to include a link embedding layer, which processes the node embeddings and turns them into link ones. The link embedding layer usually computes the inner product of the tensor, however there are other operators such as hadamard product.

   In the case of network embeddings (N2V, DW, LINE), the model returns node embeddings which have to be combined with another classification algorithm (such as logistic regression or Word2Vec).

5. The model is built, optimised with standard optimiser (Adam or SGD). The model also has a validation subset, that is used for hyper-parameter optimisation (learning rate, epoch number, batch size).

6. The output vectors of GNN indicate whether there is a link formed between two nodes.

7. After each training iteration, the performance is checked on a test set, computing Test AUC and Test Loss Metrics.

Due to the relative simplicity of the graph datasets, the graph neural networks are tremendously fast to train, a VGAE on a Cora dataset running for 400 epochs can be trained under 20 seconds (assuming GPU Cuda support). The model essentially processes a single adjacency matrix rather than a thousands of images or text vectors (Vision & NLP Tasks). The problem arises when new nodes, or links are added to the network, which requires some of the models to be re-trained, which can be a problem in large graphs of millions of nodes.

3.1.1 Datasets

The following 10 datasets have been picked for experiments. It is a mixture of datasets with varying connectivity, node amount, and network shapes. The datasets are visualised in the figure 3.1 in order to help to understand how the geometrical shapes of a given network might affect the performances of the algorithms.

3.1.2 Node Features

In order to train, GNN's do not necessarily require explicit node features of a given graph dataset, however these features often result in performance benefits. Since these features are often unavailable, there is a possibility to feed handcrafted node features into

Dataset	N (nodes)	N (Edges)	Av. Degree	Source
USAir	332	2,126	12.81	Batagelj and Mrvar (2006)
NS	1,589	2,742	3.45	Newman (2006)
PB	1,222	16,714	27.36	Ackland et al. (2005)
Yeast	2,375	11,693	9.85	Von Mering et al. (2002)
Cele	297	2,148	14.46	Watts and Strogatz (1998)
Power	4,491	6,594	2.67	Watts and Strogatz (1998)
Router	5,022	6,258	2.49	Spring et al. (2004)
Arxiv	18,722	198,110	21.10	Leskovec and Krevl (2014)
Facebook	4,039	88,234	43.69	Leskovec and Krevl (2014)
BlogCatalog	10,312	333,983	64.77	Zafarani and Liu (2009)

Table 3.1: Table showing the datasets with number of nodes, edges, average node degree and the dataset source

the models. Popular examples are topological features like node degrees or centrality measures which can be easily computed before training, or network embeddings like Node2Vec or LINE, which require greater computational time. This work does not focus on datasets with explicit node features, since it is often the case not to possess them. Instead, it is more practical to compare the GNN models against traditional heuristic approaches which do not require explicit node features as well.

3.1.3 Sampling Process

The training and testing splits are randomly resampled on every run from the graph themselves, with the ratio of 90% Train to 10% Test. To aim for better performance, the train and test splits have certain overlap between them, as a test set of one run for a particular dataset can share some edges of the test set of another run for the same dataset. This justifies why some of the runs of a particular dataset have different slopes as the rest, as the two runs are being trained on different edge-node combinations of the network.

3.1.4 Evaluation Framework

A general verdict on a model is determined not only by the highest Test AUC it has reached at some point, but as well as how it performs in the learning process during training. Here, there are several factors to evaluate. Initially, the model is checked for overfitting, ie. whether the model has reached too optimistic predictions on the training set, (measured by the training loss) and too inaccurate predictions on the test set (test loss).

If the model does not overfit, the second important factor taken into consideration are the slopes and the trends of the loss and AUC curves. The smoothness and overall trend of these curves determines the stability of the learning process, and is crucial for an

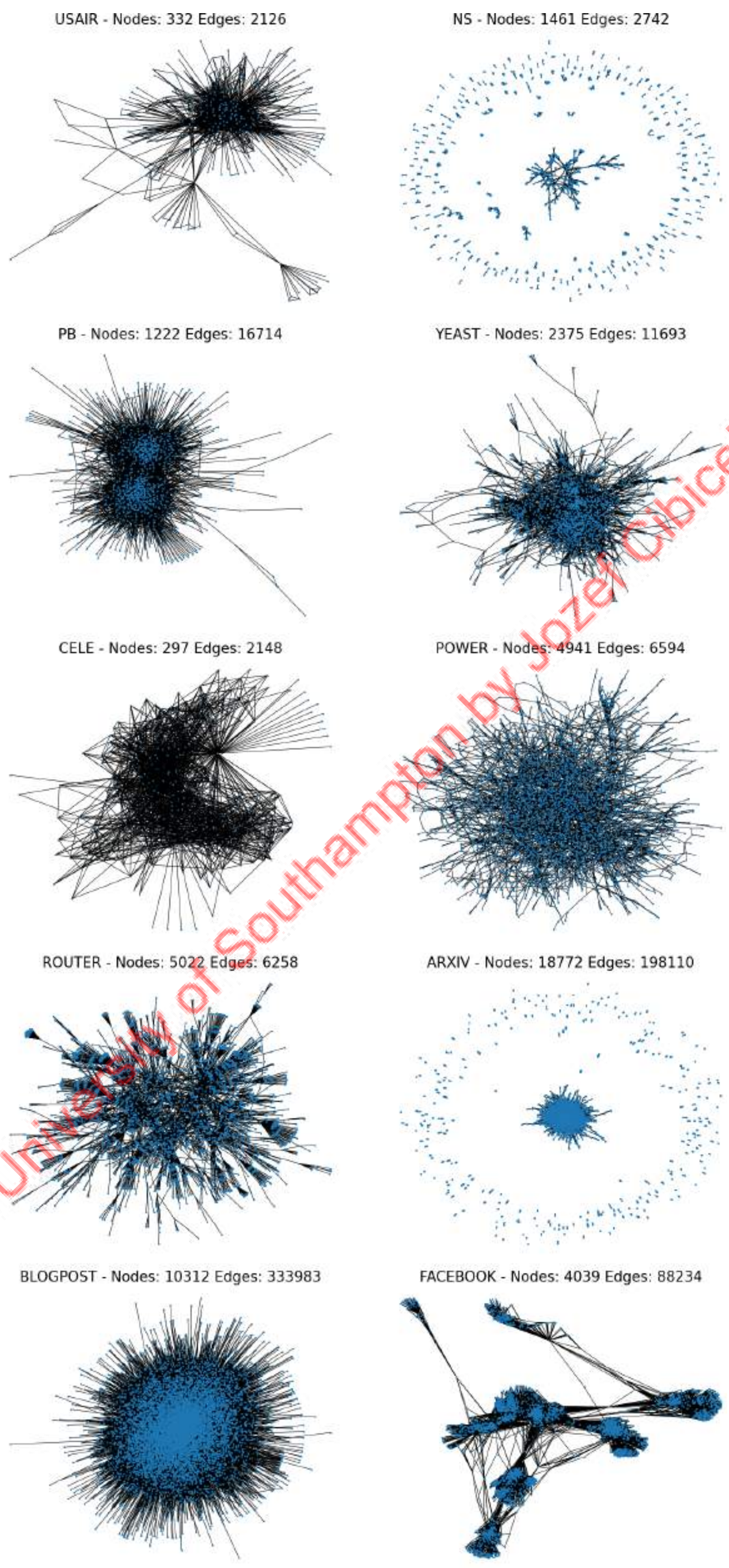

USAIR - Nodes: 332 Edges: 2126
NS - Nodes: 1461 Edges: 2742
PB - Nodes: 1222 Edges: 16714
YEAST - Nodes: 2375 Edges: 11693
CELE - Nodes: 297 Edges: 2148
POWER - Nodes: 4941 Edges: 6594
ROUTER - Nodes: 5022 Edges: 6258
ARXIV - Nodes: 18772 Edges: 198110
BLOGPOST - Nodes: 10312 Edges: 333983
FACEBOOK - Nodes: 4039 Edges: 88234

FIGURE 3.1: Datasets used throughout the experiments drawn by Python's Networkx module Hagberg et al. (2008) .

applicability of a given model. It does not matter if a model has reached perfect AUC at some point, if 10 epochs later the AUC is 50% lower and the slope has reversed.

Last but not least, a third important factor to consider is the variability between the metrics of the conducted runs. As the model is being re-trained on a different part of the network each run, it inherently captures different dynamics of the network which may result in different slopes of learning and AUC curves. A good performing model keeps this variability between the runs as low as possible or tries to minimise them with correct values of hyper-parameters.

3.2 Model Training

This section shows the performance of 5 Graph Neural Network models on the ten datasets from table 3.1 without any additional node features. Each model is run 5 times on each dataset, resulting in 250 runs and 50,000 training and testing epochs.

The documented results in the tables are computed in the following way: An argmax or argmin function is applied to the values of the each run which results in a vector V of 5 maximums or minimums which are later averaged. This is complemented with the standard deviation of these 5 values from the vector V. The figures display AUC and loss values for the training and the testing splits.

3.2.1 Parameters for all configurations

The number of epochs is 200. The number of model-specific convolutional graph layers is 2. The K of K-hop sub-sampler is 2. The batch size is 32. The dropout probability is 0.5. The anchor number is 64. The models are optimised with Adam optimiser Kingma and Ba (2014) with learning rate being 0.01. The objective function is binary cross entropy. The runs are conducted on 90% training to 10% testing splits of the edges (nodes are not split), unless noted. The size of the hidden, output and feature layers is 32. The node features are identity matrices of the graphs themselves.

Additional hyper-parameter tuning is not conducted as it is the subjectively believed by the author that neural network models should be ready to perform well out of the box in order to maintain practicality and time efficiency. The respective parameter values are recommended by the authors of the models. This also puts the GNNs and Heuristics on the equal scale, as it seems like a more fair comparison. Due to not having the need to perform hyper-parameter tuning and to use the highest possible amount of training edges for training, the validation split is not present in the runs.

3.2.2 Heuristic Methods - A Traditional Approach

Although the choice of a heuristic is subjective, a combination of popular first and second order heuristics is tested on the datasets as a baseline against the neural network models. The chosen heuristics are; Jaccard Coefficient Jaccard (1901), Preferential Attachment Zhou et al. (2009) , Adamic-Adair index Adamic and Adar (2003), and Resource Allocation Index Newman (2001). Usually, 90% of the graph is reserved for training, while randomly sampled 10% of the graph is used for predictions. The comparisons are drawn from the ROC curves which plot the ratio of false positives against the true positives.

Dataset	JACCARD	PA	AA	RA	Average perf.
ARXIV	0.81	0.95	0.89	0.94	0.87
BLOGCATALOG	0.65	0.80	0.69	0.95	0.772
CELEG	0.798	0.772	0.878	0.877	0.831
FACEBOOK	0.89	0.89	0.93	0.93	0.912
NS	0.968	0.683	0.969	0.969	0.897
PB	0.875	0.900	0.922	0.923	0.923
POWER	0.607	0.499	0.607	0.607	0.581
ROUTER	0.539	0.389	0.539	0.539	0.501
USAIR	0.906	0.891	0.949	0.953	0.924
YEAST	0.897	0.82	0.899	0.899	0.878
Average & std	0.786	0.737	0.819	0.848	0.806

TABLE 3.2: Table showing the Test AUC of the four heuristics on the 10 datasets.

The chosen heuristics perform well on the datasets, as the heuristics score on average 0.8 Test AUC. POWER and ROUTER are the only two datasets that receive poor results, as any highest AUC for these two datasets is not higher than 0.60. Overall, the best performing heuristic is Resource Allocation Index, as its average AUC is 0.848.

In addition to this, the heuristics display varying performances between the individual heuristics themselves. A typical example is CELEG, on which the AUC of the heuristics ranges from 0.77 of PA, to 0.87 of AA. This behavior is common, as the research community does not agree about a universal heuristic for all graph types and it is advised to use a mixture of heuristics for a given problem.

To combat the time complexity of the heuristics, a concurrent program utilising multiple CPU cores is written to compute the heuristic scores for ARXIV, BLOGCATALOG and FACEBOOK with speedups measured in hundreds (code included in Appendix A). This allows to bring down the overall computation time from for 4.5 hours (1 Core) FACEBOOK to 8 minutes (40 cores) for each heuristic method.

In conclusion, the heuristic approaches seem to be quick and effective on smaller to medium datasets, but can cause complications on large datasets. Combined with the fact that the power of neural networks increases as the dataset size, the next section trains the 5 GNN models on the same 10 datasets.

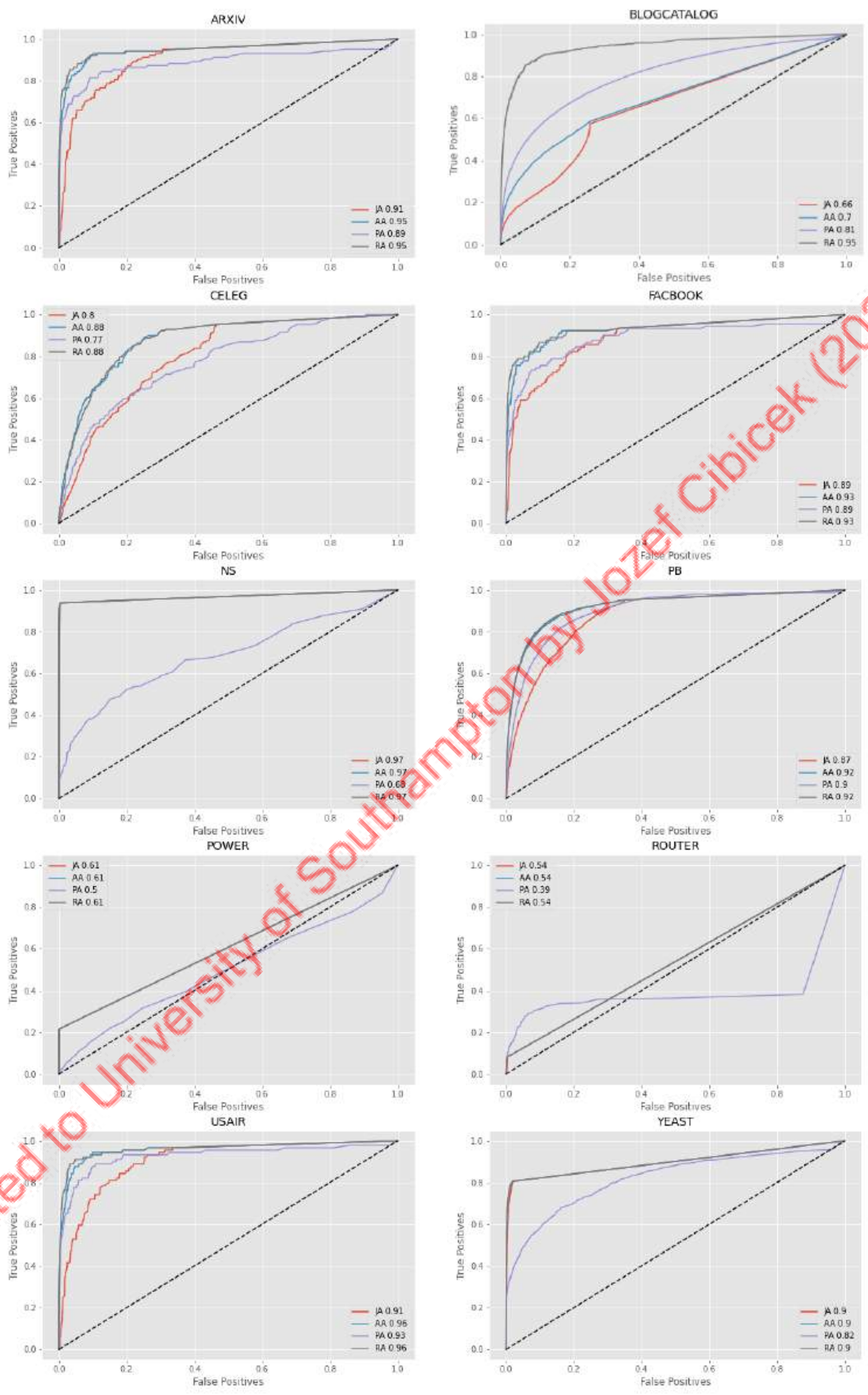


FIGURE 3.2: ROC AUC Curves of the 10 datasets for 4 Heuristics (Jaccard Coefficient, Preferential Attachment, Adamic-Adair index, and Resource Allocation Index). The black dotted line denotes AUC of 0.5, which corresponds to random guess.

3.2.3 Graph Convolutional Network (GCN) - Kipf 2016

Graph Convolutional Network can be seen as an extension to GAE, by performing graph convolutions, however, this time the extracted features are aggregated under the message passing algorithm to produce node embeddings. In this case, the GCN model is of two GCN layers, (16 channels each) and the last classification layer is an inner product of a tensor followed through one dense layer. The GCN performs well on the graph datasets

Dataset	Train AUC	Test AUC	Train Loss	Test Loss
ARXIV	0.937 ± 0.01	0.902 ± 0.008	0.527 ± 0.004	0.56 ± 0.003
BLOGCATALOG	0.831 ± 0.004	0.824 ± 0.004	0.614 ± 0.002	0.619 ± 0.002
CELEG	0.884 ± 0.006	0.806 ± 0.015	0.545 ± 0.002	0.589 ± 0.009
FACEBOOK	0.982 ± 0.001	0.980 ± 0.002	0.51 ± 0.001	0.518 ± 0.001
NS	0.998 ± 0.001	0.904 ± 0.006	0.411 ± 0.002	0.558 ± 0.007
PB	0.888 ± 0.004	0.863 ± 0.005	0.539 ± 0.002	0.566 ± 0.003
POWER	0.998 ± 0.0	0.614 ± 0.008	0.403 ± 0.001	0.677 ± 0.003
ROUTER	0.997 ± 0.0	0.615 ± 0.013	0.399 ± 0.001	0.679 ± 0.005
USAIR	0.931 ± 0.005	0.868 ± 0.013	0.507 ± 0.002	0.569 ± 0.008
YEAST	0.975 ± 0.004	0.896 ± 0.005	0.473 ± 0.002	0.56 ± 0.001
Average & std	0.942 ± 0.058	0.827 ± 0.122	0.493 ± 0.071	0.589 ± 0.053

Table 3.3: Train and Test metrics for GCN averaged over 5 runs with corresponding standard deviations.

with large number of graph nodes; (FACEBOOK, ARXIV, BLOGCATALOG) scoring 0.98, 0.90 and 0.82, Test AUC respectively, which was expected as neural networks tend to perform highly under such conditions. The lowest performing datasets; POWER and ROUTER result in very low, 0.61 Test AUC both. The other remaining datasets achieve at least 0.86 Test AUC, except the CELEG which scores 0.80 Test AUC.

Regarding the learning process of the model, more specifically, the slopes of loss and AUC curves during training, this configuration of GCN displays a variety of behaviours for different datasets. The first collection of datasets (BLOGCATALOG, FACEBOOK, PB) displays smooth, increasing AUC curves and decreasing elbow-shaped loss curves with little to no fluctuation and deviation between the runs themselves. These datasets perform just fine both from the learning perspective and with respect to their AUC curves.

The second group (ARXIV, YEAST, USAIR, CELEG), displays greater variability between their individual runs with respect to their AUC curves. Moreover, these datasets display fluctuating AUC curves, as for example shown on the AUC plots of ARXIV or YEAST. In addition to this, the model slightly overfits on USAIR and CELEG.

The third group (ROUTER, POWER, NS) have unstable and highly deviating test AUC curves. The model visibly overfits on these datasets, as both their training losses are low 0.4 while the test losses are high 0.67 (0.55 in the case of NS).

Finally, the average Test AUC for GCN is 0.827 and the average Test Loss is 0.589%. Without the underperforming ROUTER and POWER datasets, the model scores 88.037% Test AUC which is accepted as suitable for not so advanced model.

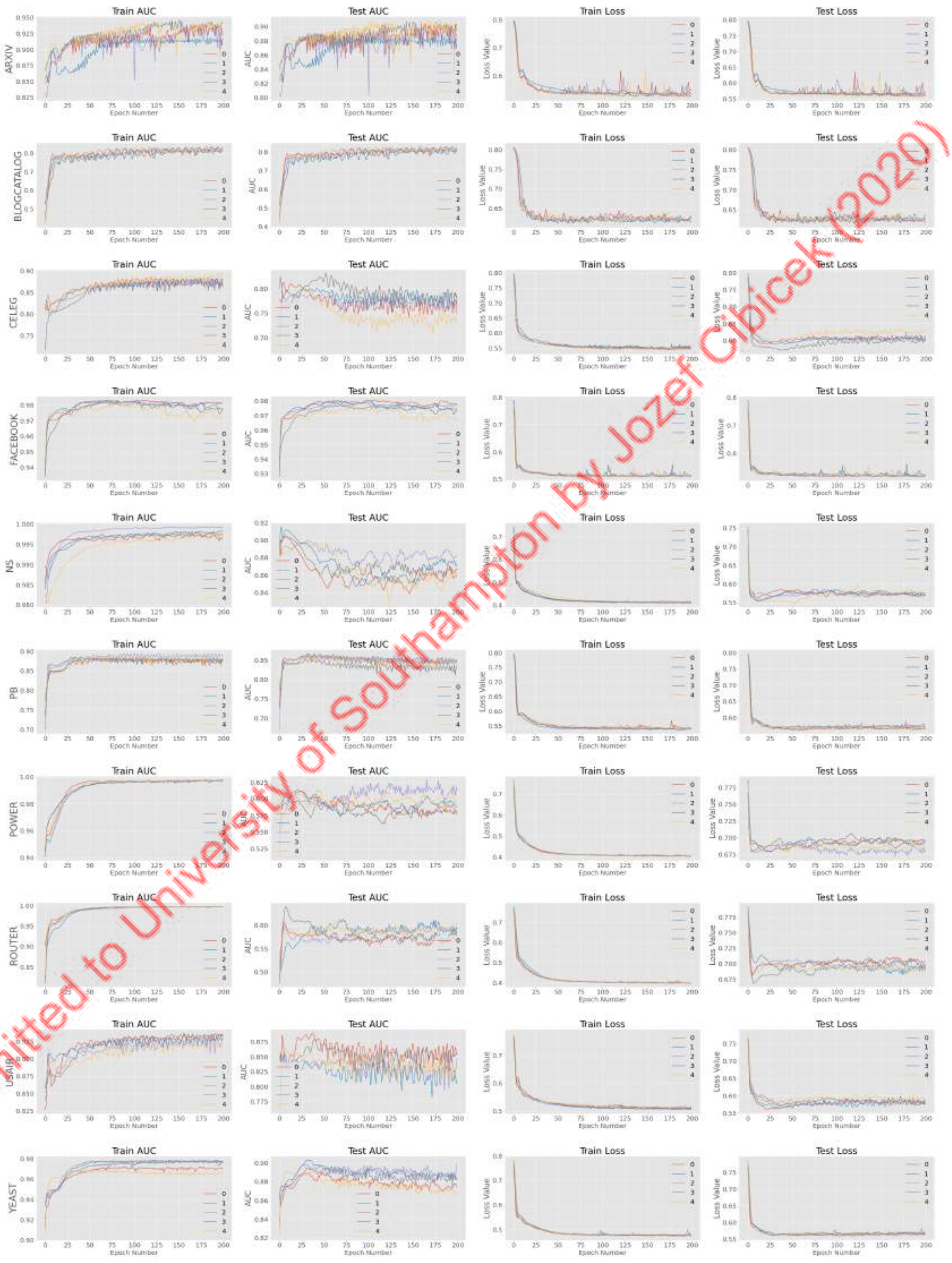


FIGURE 3.3: GCN Results of the 10 datasets with Train AUC, Test AUC, Train and Test losses.

3.2.4 GraphSAGE - Hamilton 2017

GraphSAGE utilises graph convolutions to capture embeddings which are later fed to more complex aggregators like LSTM. In this case, the output link classification layer consists of RELUs. The second model, GraphSAGE performs worse than GCN, however

Dataset	Train AUC	Test AUC	Train Loss	Test Loss
ARXIV	0.912 ± 0.012	0.871 ± 0.007	0.528 ± 0.003	0.568 ± 0.003
BLOGCATALOG	0.847 ± 0.006	0.832 ± 0.006	0.575 ± 0.004	0.596 ± 0.005
CELEG	0.889 ± 0.003	0.764 ± 0.014	0.537 ± 0.001	0.607 ± 0.008
FACEBOOK	0.955 ± 0.005	0.946 ± 0.006	0.514 ± 0.001	0.531 ± 0.001
NS	0.994 ± 0.006	0.89 ± 0.016	0.409 ± 0.006	0.571 ± 0.012
PB	0.885 ± 0.007	0.846 ± 0.006	0.533 ± 0.001	0.573 ± 0.001
POWER	0.989 ± 0.005	0.594 ± 0.016	0.41 ± 0.005	0.697 ± 0.01
ROUTER	0.986 ± 0.001	0.579 ± 0.011	0.41 ± 0.001	0.708 ± 0.006
USAIR	0.917 ± 0.012	0.845 ± 0.022	0.506 ± 0.004	0.58 ± 0.01
YEAST	0.956 ± 0.014	0.863 ± 0.009	0.474 ± 0.005	0.575 ± 0.008
Average & std	0.933 ± 0.051	0.803 ± 0.123	0.49 ± 0.061	0.601 ± 0.057

Table 3.4: Train and Test metrics for GraphSAGE averaged over 5 runs with corresponding standard deviations.

not more than 2% on average with respect to the Test AUC. This can be justified by the fact that GraphSAGE is intended to be used for large datasets of tens of thousands of nodes Hamilton et al. (2017) while the majority of the datasets in this project has below 5.000 nodes.

As expected, the model performs well on FACEBOOK, NS, ARXIV datasets, scoring 0.94, 0.89 and 0.87 Test AUC. Similarly to GCN, the model performs poorly on POWER and ROUTER datasets, scoring 0.59 and 0.57 Test AUC on average. The remaining 5 datasets have scored around 0.84 except CELEG which attained 0.76 Test AUC.

The loss and AUC curves are of different slopes compared to GCN, which displayed more conventional, ML familiar trends. In all cases, the loss plots of GraphSAGE display an elbow shaped, decreasing curvature although with occasional inconsistencies and slight variability between the individual runs (8% less Test loss variability compared to GCN).

The slopes of the AUC curves are, however a different story. Except for BLOGCATALOG, ARXIV, PB and FACEBOOK the AUC curves show a noticeable variance between the runs that can be justified by the fact that the model is being trained on a different, resampled part of the network every run. Worth mentioning are the cases of ARXIV and FACEBOOK, where the AUC curves start rapidly decreasing, but soon rise, only to be followed by a plunge down and a rise up to eventually remaining stable towards the end of the training. This is believed to be attributed to the structure of the network which has to influence the loss landscape of the optimiser, because the model does not follow the same trend on the other datasets.

In conclusion, the GraphSAGE did not perform as well as the GCN, but provided acceptable results (0.803 Test AUC and 0.601 Test Loss), combined with overfitting on some datasets (POWER, ROUTER, NS, and to some extent USAIR). This is enough to justify that GraphSAGE is not the most suitable model for performing link prediction tasks on smaller datasets.

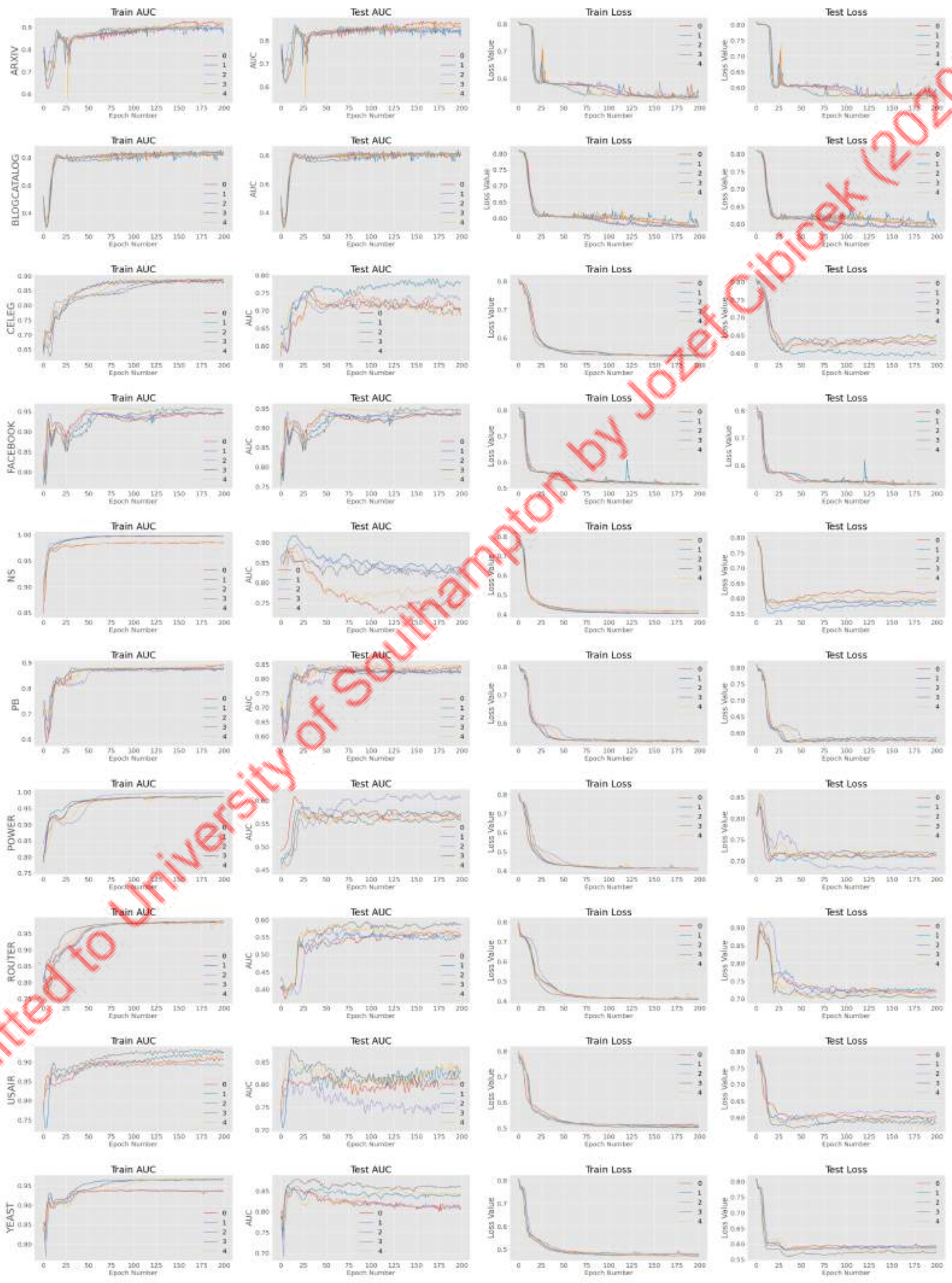


FIGURE 3.4: GraphSAGE Results of the 10 datasets with Train AUC, Test AUC, Train and Test losses.

3.2.5 Graph Attention Networks (GAT) - Veličković 2017

Graph Attention Networks implement attention layers inspired by Bahdanau et al. (2014) in which layers are stacked in order to assign different weights in a node neighbourhood by avoiding costly matrix operations.

Dataset	Train AUC	Test AUC	Train Loss	Test Loss
ARXIV	0.952 ± 0.006	0.92 ± 0.008	0.525 ± 0.002	0.555 ± 0.002
BLOGCATALOG	0.826 ± 0.022	0.808 ± 0.021	0.766 ± 0.024	0.767 ± 0.024
CELEG	0.881 ± 0.008	0.813 ± 0.006	0.547 ± 0.001	0.586 ± 0.003
FACEBOOK	0.984 ± 0.0	0.981 ± 0.001	0.511 ± 0.001	0.518 ± 0.001
NS	0.998 ± 0.001	0.907 ± 0.01	0.411 ± 0.002	0.555 ± 0.006
PB	0.887 ± 0.008	0.861 ± 0.007	0.544 ± 0.002	0.568 ± 0.004
POWER	0.998 ± 0.0	0.614 ± 0.019	0.404 ± 0.001	0.679 ± 0.01
ROUTER	0.997 ± 0.0	0.617 ± 0.007	0.4 ± 0.001	0.679 ± 0.004
USAIR	0.928 ± 0.005	0.859 ± 0.017	0.508 ± 0.004	0.574 ± 0.004
YEAST	0.978 ± 0.001	0.902 ± 0.008	0.474 ± 0.001	0.558 ± 0.007
Average % std	0.943 ± 0.06	0.828 ± 0.123	0.509 ± 0.107	0.604 ± 0.078

Table 3.5: Train and Test metrics for GAT averaged over 5 runs with corresponding standard deviations.

Regarding the test metrics, several datasets have FACEBOOK, ARXIV, YEAST reached its best performing metrics yet, Test AUC's of 0.98, 0.92, 0.90 and Test Losses of 0.518, 0.555, 0.558 respectively. Third time in a row the model performs poorly on ROUTER and POWER, 0.61 Test AUC and 0.68 Test Loss, which is becoming a standard trend among the models. The model moreover overfits in this case as the train predictions are almost 100 % correct.

Although the results of the GAT metrics look acceptable in the table above, the slopes of AUC curves on the next page help to reject this verdict. The third model brings again different trends in the learning procedure.

The loss curves follow an elbow shaped downward trend in all but the case of BLOGCATALOG, where the losses start increasing after 10th epoch which concludes visible overfitting. In addition to this, datasets like USAIR, ARXIV, YEAST, CELEG, and PB show heavy instability among AUC curves which tends to commence after around 40th epoch of the training.

Except for one case (FACEBOOK), the Test AUC curves for the remaining datasets are either decreasing, unstable or with variable runs, which rules out the applicability of GAT for link prediction unless these AUC curves become stabilised with a correct hyperparameter setting, or with an addition of a different feature matrix than the identity matrix of a graph.

Despite reaching new maximum Test AUC values for most datasets, this configuration of GAT is the worst model tested so far in terms of learning procedure, by displaying undesirable, variable and decreasing Test AUC curves.

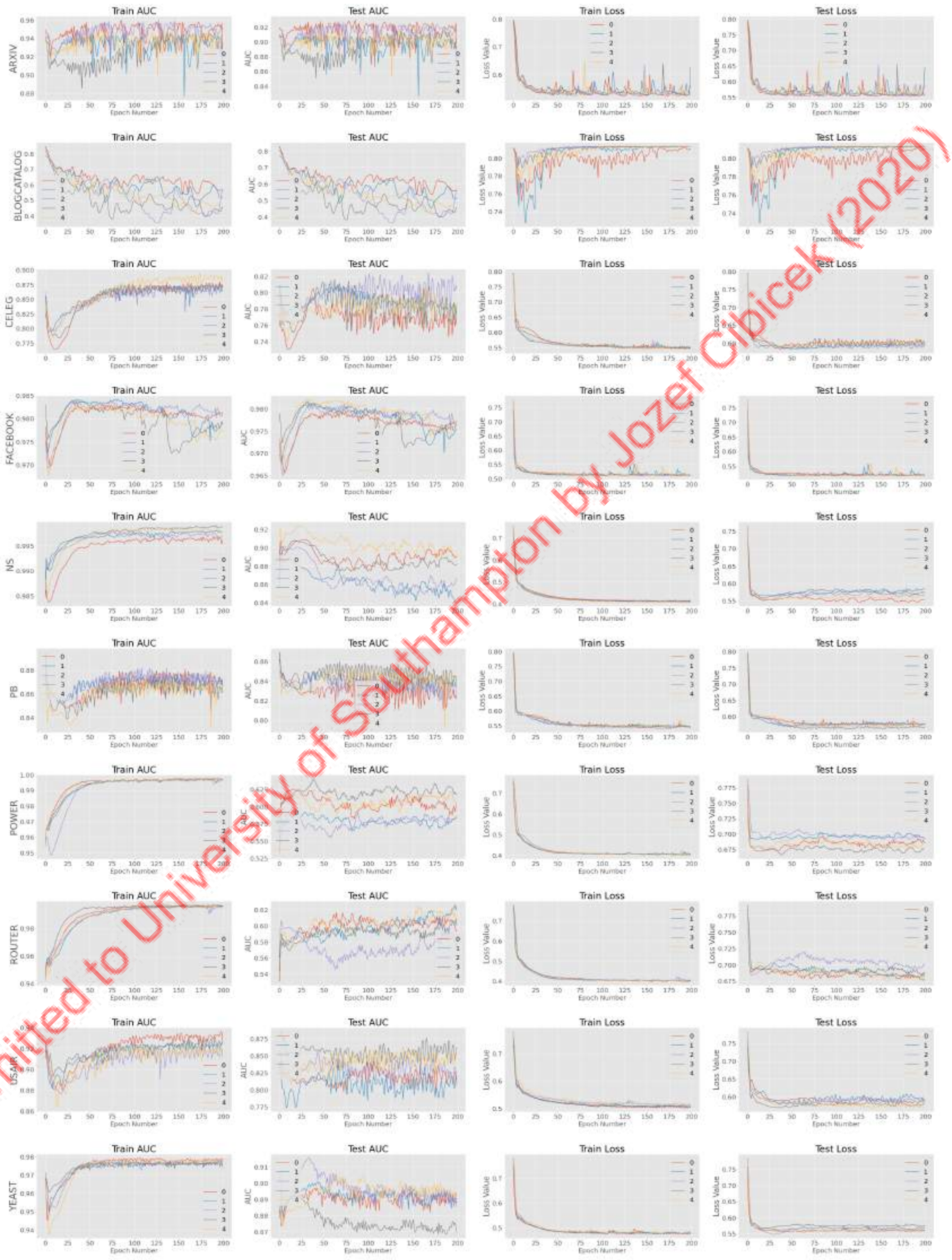


FIGURE 3.5: GAT Results of the 10 datasets with Train AUC, Test AUC, Train and Test losses.

3.2.6 Graph Isomorphism Network (GIN) - Xu et al. 2018

Graph Isomorphism Network promises the theoretical limit of how much a GNN is able to learn, by extending the Weisfeiler-Lehman test for graph isomorphism. Xu et al. (2018)

Dataset	Train AUC	Test AUC	Train Loss	Test Loss
ARXIV	0.956 ± 0.001	0.928 ± 0.001	0.527 ± 0.001	0.551 ± 0.0
BLOGCATALOG	0.758 ± 0.015	0.753 ± 0.015	0.796 ± 0.004	0.796 ± 0.004
CELEG	0.88 ± 0.005	0.804 ± 0.012	0.544 ± 0.004	0.585 ± 0.007
FACEBOOK	0.977 ± 0.001	0.975 ± 0.001	0.511 ± 0.001	0.516 ± 0.001
NS	0.997 ± 0.001	0.918 ± 0.009	0.412 ± 0.001	0.559 ± 0.006
PB	0.888 ± 0.003	0.867 ± 0.004	0.545 ± 0.001	0.566 ± 0.003
POWER	0.998 ± 0.0	0.619 ± 0.011	0.403 ± 0.001	0.677 ± 0.006
ROUTER	0.984 ± 0.006	0.594 ± 0.008	0.412 ± 0.006	0.699 ± 0.006
USAIR	0.924 ± 0.004	0.869 ± 0.023	0.512 ± 0.003	0.57 ± 0.014
YEAST	0.974 ± 0.002	0.902 ± 0.004	0.476 ± 0.001	0.558 ± 0.003
Average & std	0.934 ± 0.075	0.823 ± 0.13	0.514 ± 0.114	0.608 ± 0.087

Table 3.6: Train and Test metrics for GIN averaged over 5 runs with corresponding standard deviations.

GIN has achieved lower aggregate test metrics than the previous GAT (Test AUC: 0.823 against 0.828, Test Loss: 0.608 against 0.604), nevertheless it provides better conditions during the training of the model and therefore higher degree of applicability.

The model classically overfits on POWER and ROUTER, that had again reached low test metrics (0.61, 0.59 Test AUC and 0.677, 0.699 Test Loss). Except for ARXIV, FACEBOOK, CELEG, USAIR, and PB, the model performs unacceptably on all datasets, by displaying decreasing AUC curves or too much run-variability or run-instability. If the model follows an early stopping scenario during training, and the training is stopped after the Test AUC reaches a particular maximum value, the decreasing trends of the Test AUC can be avoided.

The learning process again displays some unusual learning trends, such as the case of BLOGCATALOG, where a large dataset has achieved poor model performance combined with overfitting of the model, a scenario not yet encountered. Compared to GCN and GAT, both loss and AUC curves of POWER and ROUTER follow a standard decreasing or increasing slope respectively, although the model still overfits. FACEBOOK is a nice example of the unstable property of this model, as few runs follow standard AUC shape, whereas the other half of the runs is unstable with inconsistencies. This dataset typically follows a normal development during training, as it belongs to the group of datasets with high node degree.

GIN is a model with highest standard deviation among the observed metrics 0.075, 0.13, 0.114, 0.087, which means the performance of the model is very sensitive to the part of network on which the model is being trained on. This is disappointing as the authors describe the model as the state of the art convolutional GNN. GIN proves to be a better alternative to the GAT above, as the model overfits on less datasets, but is not a better alternative than GraphSAGE or GCN.

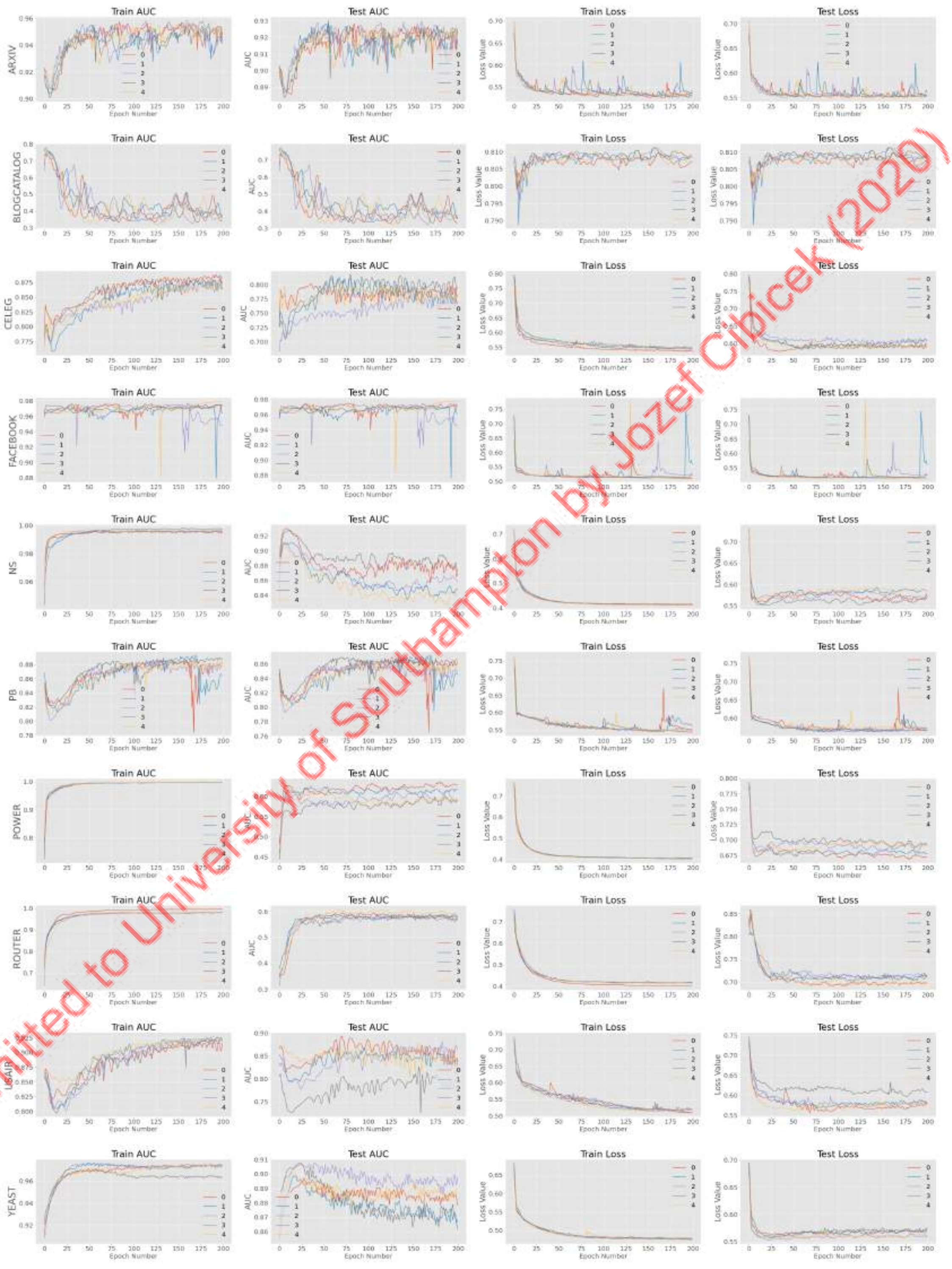


FIGURE 3.6: GIN Results of the 10 datasets with Train AUC, Test AUC, Train and Test losses.

3.2.7 P-GNN - You et al. 2019

PGNN is the most recent model to be tested, incorporating position aware graph embeddings. This time the PGNN uses a novel parameter called 'anchor set', which determines the size of anchors in the dataset. An anchor set is a k-random subset of nodes from a given network that is meant as a reference point while computing the position aware embeddings for each node. The PGNN follows the well known trends of the previous

Dataset	Train AUC	Test AUC	Train Loss	Test Loss
ARXIV	0.975 ± 0.005	0.935 ± 0.005	0.557 ± 0.01	0.581 ± 0.003
BLOGCATALOG	0.892 ± 0.016	0.868 ± 0.017	0.579 ± 0.009	0.592 ± 0.009
CELEG	0.855 ± 0.01	0.723 ± 0.014	0.595 ± 0.003	0.641 ± 0.015
FACEBOOK	0.955 ± 0.003	0.937 ± 0.002	0.551 ± 0.007	0.557 ± 0.003
NS	0.991 ± 0.002	0.922 ± 0.011	0.47 ± 0.003	0.566 ± 0.004
PB	0.903 ± 0.021	0.855 ± 0.021	0.571 ± 0.001	0.601 ± 0.004
POWER	0.975 ± 0.003	0.593 ± 0.019	0.474 ± 0.003	0.677 ± 0.003
ROUTER	0.961 ± 0.002	0.586 ± 0.035	0.475 ± 0.002	0.698 ± 0.008
USAIR	0.912 ± 0.014	0.831 ± 0.02	0.542 ± 0.003	0.6 ± 0.008
YEAST	0.976 ± 0.002	0.89 ± 0.006	0.519 ± 0.003	0.594 ± 0.001
Average & std	0.94 ± 0.046	0.814 ± 0.134	0.533 ± 0.046	0.611 ± 0.047

TABLE 3.7: Train and Test metrics for PGNN averaged over 5 runs with corresponding standard deviations.

models, by performing well on datasets with higher node degrees (ARXIV 93%, FACEBOOK 94%, NS 92% Test AUC) but this time it reaches a significant increase in terms of Test Metrics on 3 datasets (ARXIV, BLOGCATALOG, NS)

More specifically, BLOGCATALOG had achieved its highest Test AUC 0.86 and its lowest Test Loss; 0.592, an increase of 4% and decrease of 0.020. The absolute maximum has reached ARXIV, a significant increase of 3% Test AUC since the maximum reached by GCN. YEAST has taken its best performance yet from the learning perspective, as both its learning and AUC curves became stabilised and reached 0.89 Test AUC coupled with 0.594 Test Loss. The worst performing dataset is again ROUTER which scored 0.58 Test AUC, combined with the highest standard deviation of Test AUC among its runs (0.035).

The slopes of the loss curves are again different, but this time they do not follow unacceptable slopes or decreasing trends as in the case of GAT or GIN. The model also does not display a significant amount of variability or instability between the runs, except for CELEG, or to some extent USAIR. YEAST had noted its best performance out of all models, as the both AUC curves became stabilised, and do not decrease as it used to be the case in previous models.

Overall, the PGNN performed very well, with couple of new best metrics for (POWER, ARXIV, NS, BLOGCATALOG). It moreover achieved the lowest variability among the main metrics, most importantly the lowest test loss variability from all models; 0.047.

PGNN appears to be one of the best performing models yet, along with GCN and GraphSAGE.

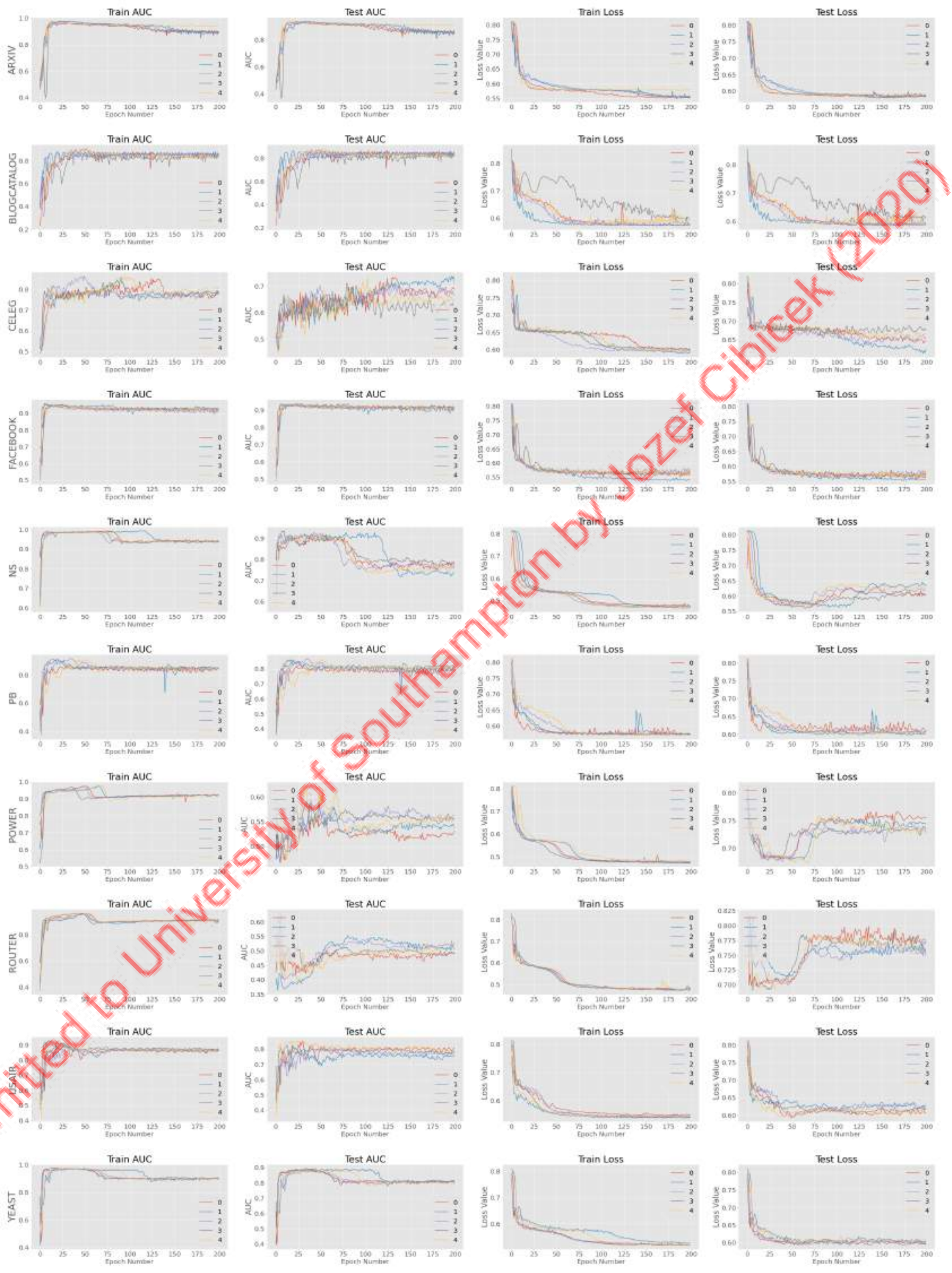


FIGURE 3.7: PGNN Results of the 10 datasets with Train AUC, Test AUC, Train and Test losses.

3.2.8 Summary of the Performances

The 5 tested models (GCN, GraphSAGE, GAT, GIN, PGNN) perform with various success on the ten datasets. This was anticipated as all 5 models have different architecture and capture the graph datasets in individual manner. The best performing models are GCN, PGNN and GraphSAGE as GAT displays unstable AUC curves during training. Among the 5 models, GIN can be considered as mediocre model, due to being unstable, but better than GAT and worse than SAGE.

Overall, the models have done an acceptable job at predicting links, scoring 0.819 Test AUC on average, considering there was not any supplied feature information, making the models operating solely with identity matrices of the graphs.

3.2.9 Dataset Findings

According to the results, the datasets can be classified to three groups by their average node degree connectivity. It is visible that as the average node degree was rising, the models had less problems operating on these datasets and displayed smooth, well performing loss and AUC curves, with less dispersion among the conducted runs. This is explainable due to the low connectivity of the datasets, measured by average node degree of a dataset (Dataset Table), which means the lower the average node degree of a graph, the harder it is to perform link prediction successfully. In other words, the less links are present between nodes in a graph, the less information the algorithm has for inference of new links.

Group 1 (FACEBOOK, BLOGCATALOG, PB, ARXIV) except for 2 cases (BLOGCATALOG at GAT, GIN) displays smooth, decreasing learning curves and favourable test AUC metrics. These datasets usually account for above 0.90 Test AUC and 0.5 Test Loss. The average node degree of these four datasets is 39.23.

Group 2. (YEAST, USAIR, CELEG) tends to perform just fine, with around 0.85 Test AUC, although there are cases where these datasets show variable metrics, and even decreasing AUC's curves after some point in the training. The average node degree for these datasets is 12.37.

Group 3. (ROUTER, POWER, NS) usually makes the models overfit, and except one case (PGNN), remains a challenge for the GNN models. This must be explained by their low (2.87) average node degree that fails the models to succeed at reaching high test AUC's and smooth learning curves.

What is also interesting, that in some cases, the model's learning capabilities improve very little, if at all after around 100th epoch, which could eliminate the need for training the models for greater number of epochs and therefore cut development time.

Dataset	Train AUC	Test AUC	Train Loss	Test Loss
ARXIV	0.946 ± 0.021	0.911 ± 0.023	0.533 ± 0.012	0.563 ± 0.011
BLOGCATALOG	0.831 ± 0.043	0.817 ± 0.038	0.666 ± 0.095	0.674 ± 0.089
CELEG	0.878 ± 0.012	0.782 ± 0.034	0.554 ± 0.021	0.602 ± 0.021
FACEBOOK	0.971 ± 0.013	0.964 ± 0.018	0.519 ± 0.016	0.528 ± 0.015
NS	0.996 ± 0.003	0.908 ± 0.011	0.423 ± 0.024	0.562 ± 0.006
PB	0.89 ± 0.006	0.858 ± 0.007	0.546 ± 0.013	0.575 ± 0.013
POWER	0.992 ± 0.009	0.607 ± 0.011	0.419 ± 0.028	0.681 ± 0.008
ROUTER	0.985 ± 0.013	0.598 ± 0.015	0.419 ± 0.028	0.693 ± 0.012
USAIR	0.922 ± 0.007	0.854 ± 0.015	0.515 ± 0.014	0.579 ± 0.011
YEAST	0.972 ± 0.008	0.891 ± 0.014	0.483 ± 0.018	0.569 ± 0.014
Average & std	0.938 ± 0.053	0.819 ± 0.118	0.507 ± 0.072	0.602 ± 0.055

Table 3.8: The table shows arithmetic average of all observed metrics for all datasets.

Model Name	Train AUC	Test AUC	Train Loss	Test Loss
GCN	0.942 ± 0.058	0.827 ± 0.122	0.493 ± 0.071	0.589 ± 0.053
SAGE	0.933 ± 0.051	0.803 ± 0.123	0.493 ± 0.061	0.601 ± 0.057
GAT	0.943 ± 0.061	0.828 ± 0.123	0.509 ± 0.107	0.604 ± 0.078
GIN	0.934 ± 0.075	0.823 ± 0.131	0.514 ± 0.114	0.608 ± 0.087
PGNN	0.941 ± 0.046	0.814 ± 0.134	0.533 ± 0.046	0.611 ± 0.047
Average & std	0.938 ± 0.004	0.819 ± 0.009	0.5077 ± 0.015	0.602 ± 0.007

Table 3.9: The table shows arithmetic average of all observed metrics for all models.

3.2.10 Heuristics Against GNNs

The 4 heuristics attain very similar performance compared to that of Graph Neural Network models. To be more specific, 0.806 Test AUC compared to the 0.819 achieved by the GNNs. There are also cases (POWER and ROUTER) where the heuristic approaches imitate the performances of featureless GNNs as they score almost identical Test AUC values.

Regarding practicality, heuristics seem to be a simple and easy to use method for computing the predictions but they lose this status as the graphs grow in size. The observed variability between the methods themselves also arises confusion on which method to use as different heuristics utilise different assumptions about the graph types. Unless they utilise parallel processing or GPU acceleration, GNNs beat these approaches in time efficiency as the computation time increases from minutes to several hours.

3.2.11 Final Remarks

It would be interesting to see performing the GNN models with explicit node features, as recommended by the authors. This is the focus of the following section, to explore and find a set of node features that significantly improve the performances of the models, while being simple and not computationally expensive to be computed.

3.3 Supply of Node Features to Models

This section attempts to improve the performance of the GNN models with the addition of explicit node features to their training. Two approaches are explored in this work namely;

1. SEAL - State of the art architecture for Link Prediction on graphs published by Zhang and Chen (2018). The framework is tested on the same 10 datasets as above for 100 epochs and 5 different runs.
2. Generation of Graphlet Node Signatures Milenković and Pržulj (2008) and their application as explicit node features in the training of the previous 5 GNN models on the same 10 datasets as their featureless version.

3.3.1 Motivation and Introduction

The frequent lack of explicit node features combined with the impracticality of using further latent algorithms like Node2Vec or LINE in order to generate them, raises a question asking how is it possible to improve the GNN training process while not significantly increasing a time complexity of the overall process. A most practical way would be to find a particular set of features that can be computed for each graph before the training of the GNNs, which is also called feature hand-crafting. The subject of finding the most accurate node features while preserving time efficiency remains a research topic on its own. At the moment, there are several ways how to generate own node features.

1. Node Embedding Algorithms - Node2Vec Grover and Leskovec (2016), LINE Tang et al. (2015), DeepWalk Perozzi et al. (2014). These algorithms provide accurate embeddings of the graph, however they require a solid amount of computational resources and therefore time which is already reserved by the training of the GNN models.
2. Node Centralities & Degrees - Degree measures, Eigenvector centrality, Katz centrality. Centrality measures provide almost instant computation of node information, however there is a great number of centrality measures leading to multitude of choices (similar scenario to choice of a 'correct' heuristics). These features also do not capture the overall importance of the node to such high extent as the algorithms above.
3. Other approaches - An interesting approach has been proposed in Liu et al. (2017) to train a Generative Adversarial Network to learn these features, however this approach is suspected to suffer from the same time complexities as the Node Embedding Algorithms.

3.3.2 SEAL and DCGN - Zhang and Chen 2018

SEAL and DCGNN are ran on the datasets with the exception of node features, which are in this case random vectors of uniformly distributed numbers. As SEAL does not rely on explicit node features, it automatically precomputes its own explicit features from the random dummy ones. The SEAL framework is runing atop of DCGN, a GNN architecture published in Zhang et al. (2018). The SEAL framework is itself flexible and it can be used with any GNN model instead of DCGN.

The number of SEAL layers is 3, k = 0.6. The batch size is 32. Except for ARXIV, BLOGPOST and FACEBOOK, the number of hops which the SEAL feature extractor takes into account for each node is 2. Larger values of this parameter exponentially increase the memory consumption and lead to OOM for the first three datasets.

Dataset	Train AUC	Test AUC	Train Loss	Test Loss
ARXIV	0.999 ± 0.0	0.998 ± 0.0	0.033 ± 0.002	0.038 ± 0.002
BLOGCATALOG	0.973 ± 0.002	0.963 ± 0.001	0.218 ± 0.006	0.246 ±0.003
CELEG	0.935 ± 0.011	0.892 ± 0.006	0.34 ± 0.026	0.428 ± 0.018
FACEBOOK	0.998 ± 0.0	0.996 ± 0.0	0.057 ± 0.003	0.068 ± 0.004
NS	0.993 ± 0.001	0.985 ± 0.004	0.093 ± 0.006	0.125 ± 0.02
PB	0.975 ± 0.004	0.949 ± 0.002	0.219 ± 0.011	0.291 ± 0.006
POWER	0.885 ± 0.007	0.875 ± 0.009	0.421 ± 0.01	0.438 ± 0.012
ROUTE	0.972 ± 0.005	0.963 ± 0.007	0.218 ± 0.018	0.244 ± 0.023
USAIR	0.975 ± 0.001	0.971 ± 0.002	0.207 ± 0.004	0.223 ± 0.011
YEAST	0.989 ± 0.002	0.983 ± 0.002	0.132 ± 0.012	0.152 ± 0.008
Average & std	0.969 ± 0.035	0.957 ± 0.042	0.194 ± 0.122	0.225 ± 0.136

TABLE 3.10: Train and Test metrics for SEAL averaged over 5 runs with corresponding standard deviations.

The results just confirm those published by the authors in Zhang and Chen (2018). Contradicting to the other models, POWER and ROUTER score significantly higher Test AUC values, on average 0.20 higher than the other models. Due to memory limitations, the BLOGCATALOG dataset is downsampled to 6000 nodes and 120,695 edges which is less than half the number of nodes and edges from its original version. Despite that, SEAL performs attains 0.963 Test AUC on this dataset. Overall, this architecture combination scores almost perfect scores, 0.957 Test AUC and 0.223 Test loss, however with some objections.

SEAL should be taken into account more as an feature extraction method than a GNN approach to Link Prediction as it precomputes its own features. This feature precomputation can be very time consuming on larger datasets, and is definitely the most time consuming model tested. For example to compute the datasets for SEAL on the ARXIV dataset, the feature extraction took 20 minutes and running the DCGN almost 100, considering the model has been ran for 100 epochs. The model can be stopped after approximately 50 epochs as it sometimes overfits (PB) and mostly does not bring any significant improvement afterwards (not to mention the training takes 1/2 of the time). It is definitely a superior approach on smaller to medium graphs where other heuristics / GNNs fail to achieve high scores, but should be not considered on large graphs due to its heavy resource use.

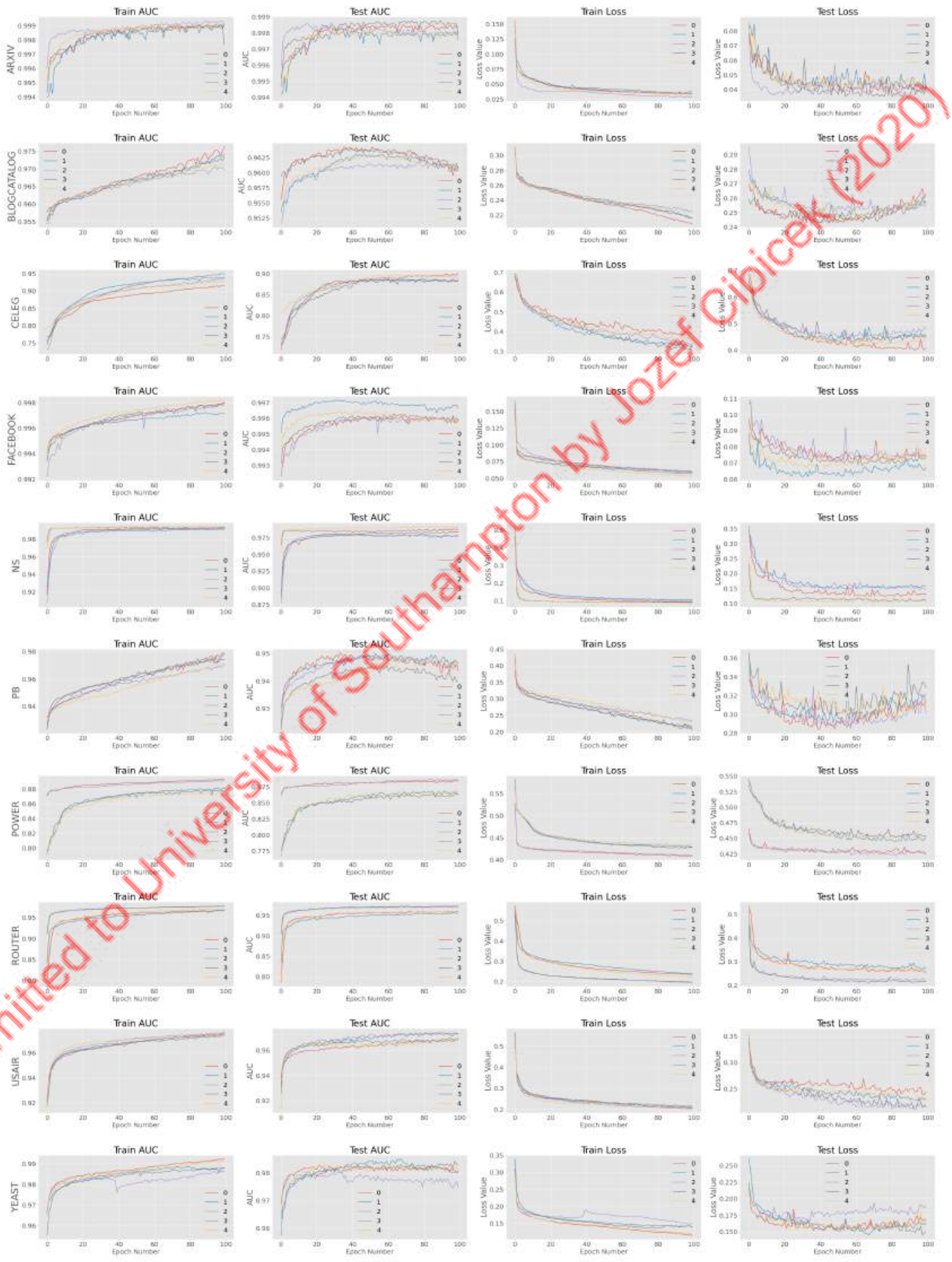


FIGURE 3.8: SEAL Results of the 10 datasets with Train AUC, Test AUC, Train and Test losses.

3.3.3 Graphlet Degree Signatures

Graphlet Degree Signatures, invented by Milenković and Pržulj (2008) are essentially a generalisation of node degrees, which is a common topological feature describing a node of a graph. Contrary to a node degree, which counts the number of connected edges to a node, Graphlet degree, counts the number of graphlets connected to this node. It promises to capture a wider information of a node topology into its 73 dimensional vector, called the orbit vector. A graphlet is small connected non-isomorphic induced subgraphs of a large network Pržulj et al. (2004). There are 29 types of graphlets, as illustrated in the figure below.

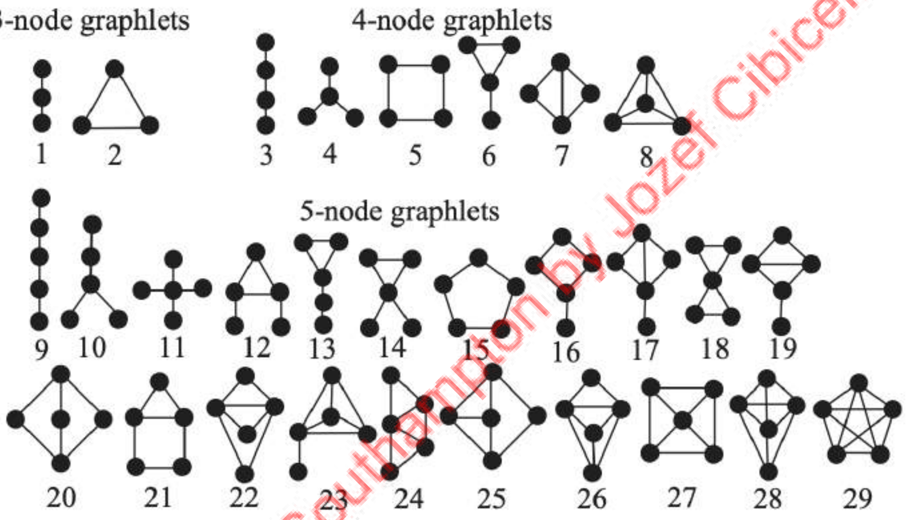


FIGURE 3.9: List of all 3, 4 and 5 node graphlets Pržulj et al. (2004). The remaining two node graphlet is missing in this figure.

The ten datasets are now passed to the orbit counting algorithm, which generates the 73 long weighted degree vector for each node of the graph. These vectors are then combined into node feature matrix X which is then passed to the GNN models. For optimisation purposes, the actual implementation used for computing the orbital vectors of the datasets is of Hocevar (2016).

The features take some time to be generated, due to the 333,000 number of edges the compiled C++ binaries ran for 11 hours in the case of BLOGCATALOG, whereas the computation for lower to medium sized graphs occurred within acceptable time frame (seconds to several minutes). This time should be added to overall training time of the models, as the models need these features in order to be run.

The 60 self-loops of ARXIV are removed from the dataset as the self-loops violate the requirements of the orbit counting algorithm. Other than that, the parameter settings remained as before. The subsequent experiment consists of running all models from the previous run except SEAL, as the model has been already tested on its own computed features in the first run.

3.3.4 Results and Discussion

The models greatly benefit from the additional features. The 73 dimensional orbital feature vectors have improved the test metrics, the average value of Test AUC has increased by 4 points, in the case of GraphSAGE as high as by 7 points.

Speaking of individual datasets, the most observable changes are in the case of POWER and ROUTER which have increased their Test AUC performance from 0.6 to 0.85, a significant 0.25 AUC increase. The additional features have also changed the way how the models learn, as the gradient of the loss curves became less steep, and more constant over the course of the training. The AUC and loss curves display significantly less variability among the runs, supported by the fact that the average standard deviation has decreased from the previous run from 0.118 to 0.040 and from 0.055 to 0.036 respectively.

Nonetheless, the additional orbital features have not improved the performance of all the models, as some particular datasets attained lowered performance compared to the run without the features. This is observable in the case of GAT, in which except for ROUTER, all the datasets scored higher metric values in the first run coupled with observable disastrous curve slopes in the second run. (0.5 difference in Test AUC).

Dataset	Train AUC	Test AUC	Train Loss	Test Loss
ARXIV	0.944 ± 0.026	0.91 ± 0.027	0.533 ± 0.01	0.563 ± 0.011
BLOGCATALOG	0.804 ± 0.057	0.801 ± 0.057	0.646 ± 0.061	0.648 ± 0.061
CELEG	0.783 ± 0.059	0.748 ± 0.056	0.617 ± 0.054	0.63 ± 0.05
FACEBOOK	0.948 ± 0.024	0.945 ± 0.025	0.545 ± 0.038	0.546 ± 0.038
NS	0.965 ± 0.005	0.952 ± 0.012	0.505 ± 0.014	0.525 ± 0.008
PB	0.846 ± 0.048	0.838 ± 0.051	0.584 ± 0.034	0.587 ± 0.034
POWER	0.952 ± 0.021	0.834 ± 0.013	0.531 ± 0.015	0.585 ± 0.005
ROUTER	0.914 ± 0.052	0.757 ± 0.089	0.557 ± 0.037	0.622 ± 0.046
USAIR	0.863 ± 0.043	0.833 ± 0.043	0.578 ± 0.032	0.593 ± 0.027
YEAST	0.931 ± 0.037	0.893 ± 0.032	0.547 ± 0.026	0.565 ± 0.02
Average & std	0.895 ± 0.062	0.851± 0.068	0.564 ± 0.040	0.586 ± 0.036

TABLE 3.11: The table shows arithmetic average of all observed metrics for all datasets with supplied features.

Model Name	Train AUC	Test AUC	Train Loss	Test Loss
GCN	0.915 ± 0.059	0.872 ± 0.063	0.545 ± 0.04	0.571 ± 0.035
GraphSAGE	0.917 ± 0.055	0.876 ± 0.051	0.537 ± 0.028	0.563 ± 0.026
GAT	0.841 ± 0.105	0.792 ± 0.124	0.614 ± 0.085	0.637 ± 0.073
GIN	0.897 ± 0.082	0.849 ± 0.089	0.556 ± 0.053	0.581 ± 0.054
PGNN	0.907 ± 0.055	0.866 ± 0.064	0.568 ± 0.026	0.582 ± 0.026
Average & std	0.895 ± 0.028	0.851 ± 0.031	0.564 ± 0.027	0.586 ± 0.026
SEAL	0.969 ± 0.035	0.957 ± 0.042	0.194 ± 0.122	0.225 ± 0.136

TABLE 3.12: The table shows arithmetic average of all observed metrics for all models with supplied features

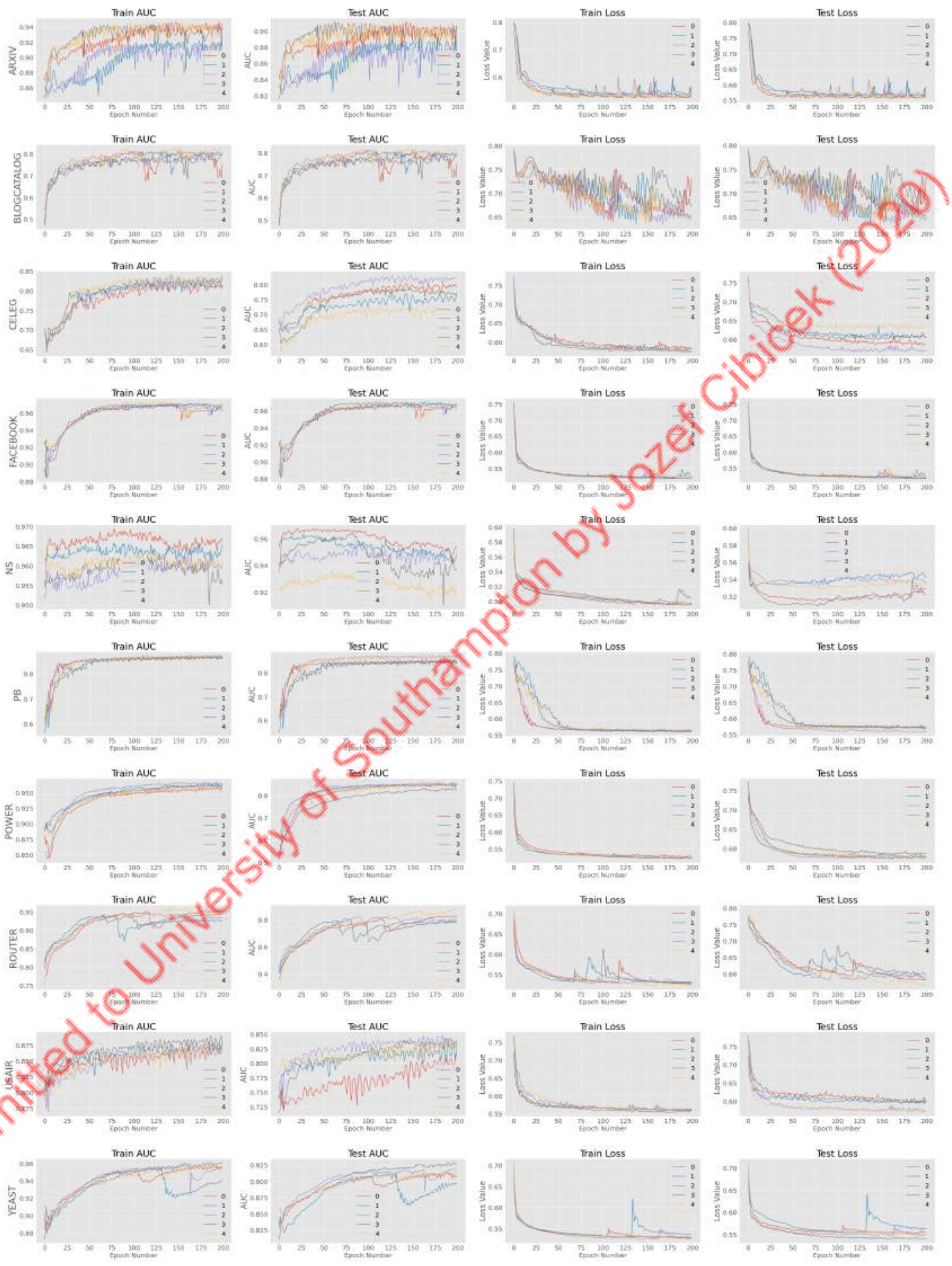


FIGURE 3.10: Results of GCN running on all 10 datasets with supplied feature matrix containing orbital signatures.

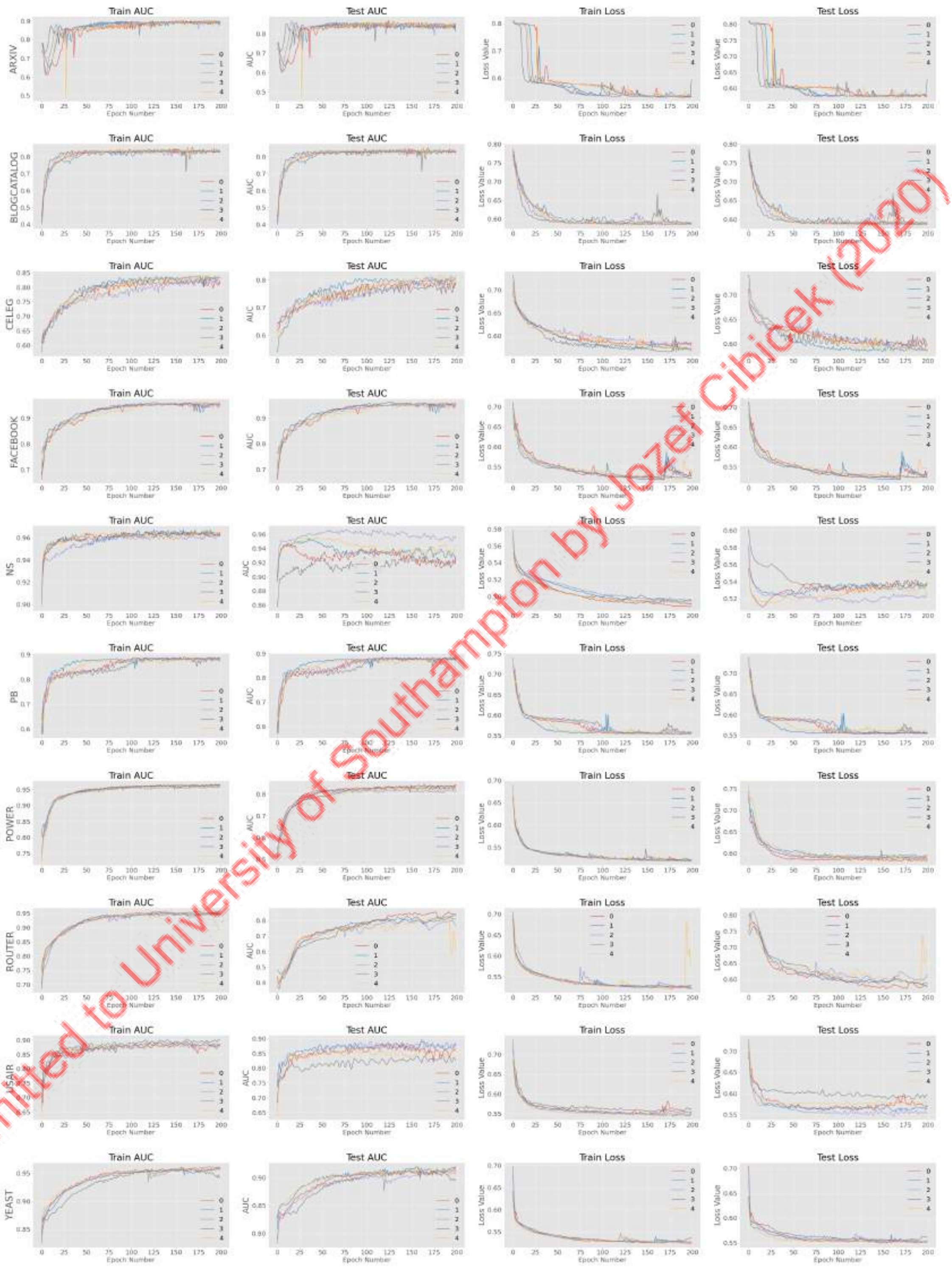


FIGURE 3.11: Results of SAGE running on all 10 datasets with supplied feature matrix containing orbital signatures.

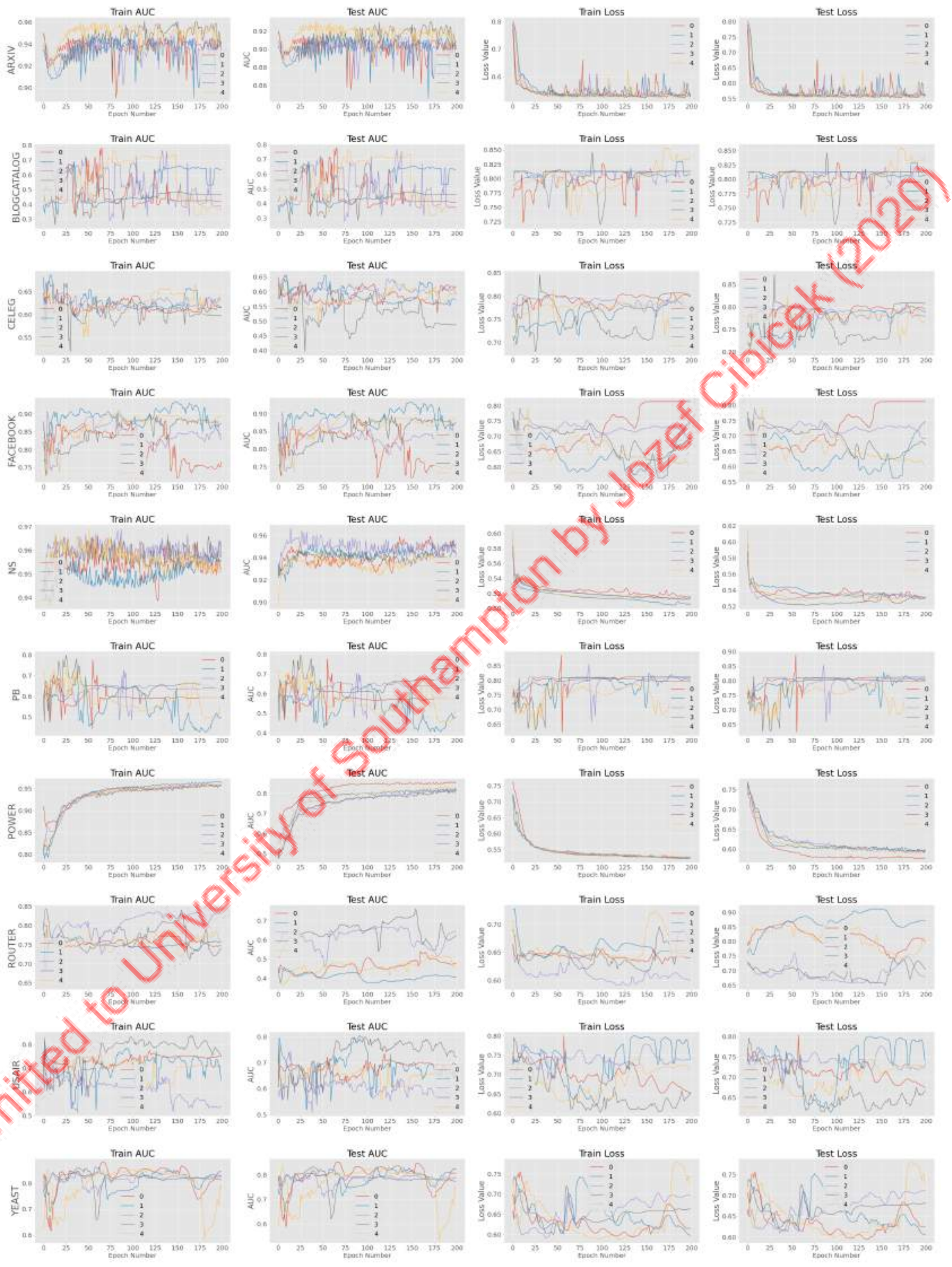


FIGURE 3.12: Results of GAT running on all 10 datasets with supplied feature matrix containing orbital signatures.

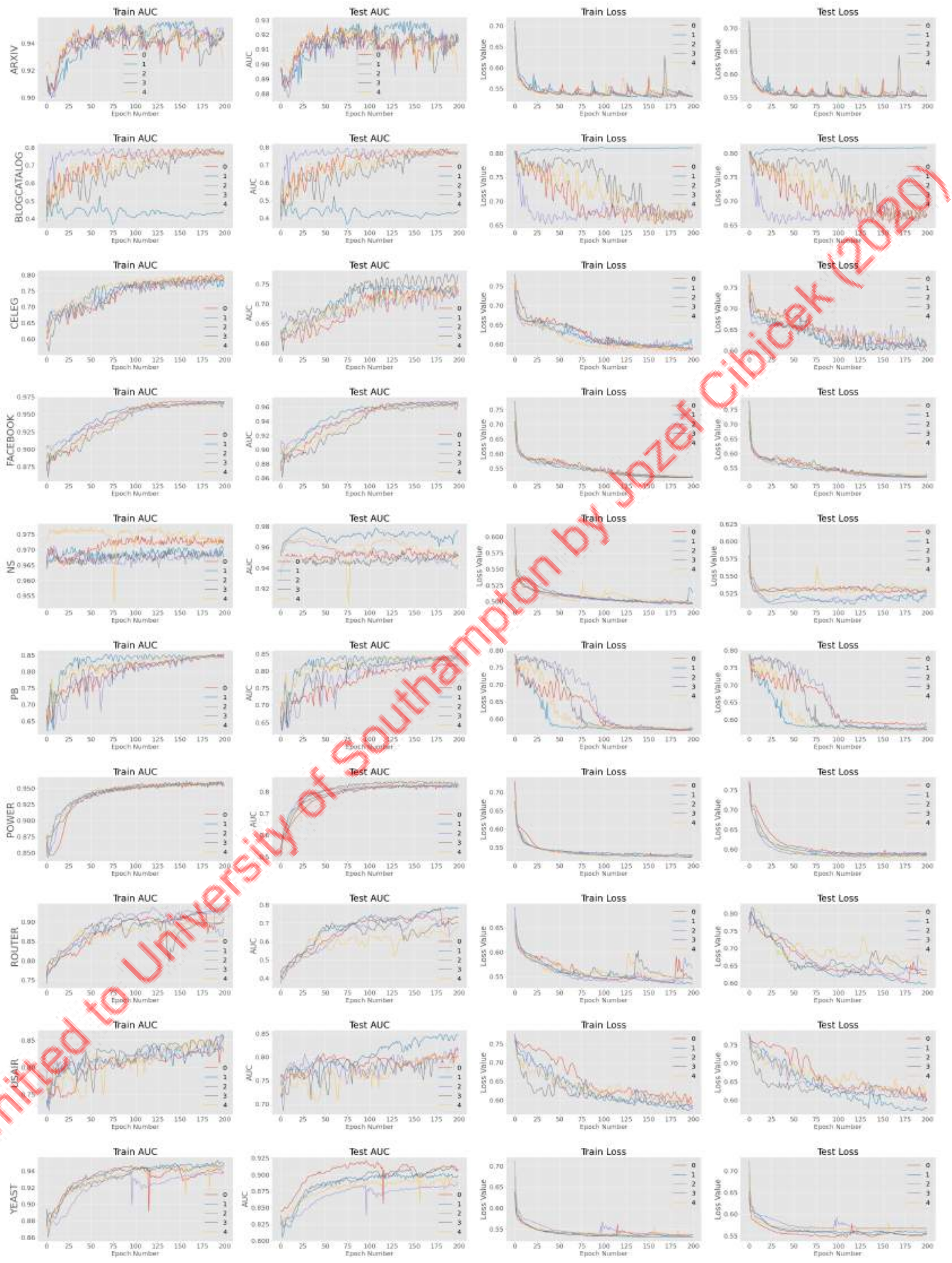


FIGURE 3.13: Results of GIN running on all 10 datasets with supplied feature matrix containing orbital signatures.

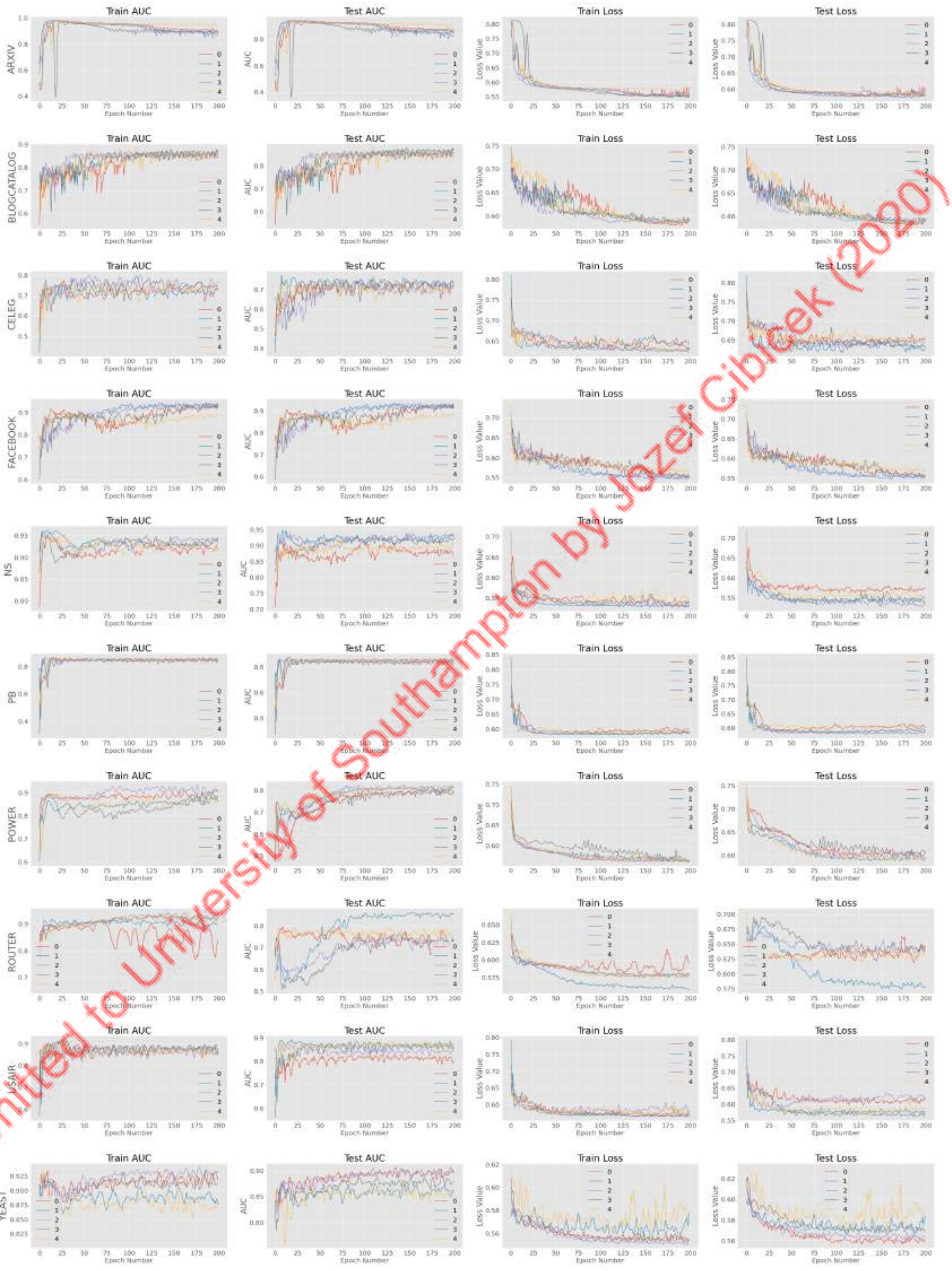


FIGURE 3.14: Results of PGNN running on all 10 datasets with supplied feature matrix containing orbital signatures.

Dataset	Train AUC	Test AUC	Train Loss	Test Loss
ARXIV	0.936+-0.012	0.902+-0.011	0.527+-0.005	0.56+-0.004
BLOGCATALOG	0.816 ± 0.005	0.811 ± 0.005	0.636 ± 0.003	0.638 ± 0.004
CELEG	0.831 ± 0.007	0.784 ± 0.035	0.578 ± 0.003	0.598 ± 0.02
FACEBOOK	0.972 ± 0.001	0.97 ± 0.001	0.518 ± 0.001	0.519 ± 0.001
NS	0.965 ± 0.003	0.958 ± 0.011	0.496 ± 0.001	0.524 ± 0.01
PB	0.873 ± 0.001	0.861 ± 0.009	0.562 ± 0.002	0.568 ± 0.005
POWER	0.964 ± 0.003	0.853 ± 0.01	0.523 ± 0.002	0.577 ± 0.004
ROUTER	0.95 ± 0.007	0.827 ± 0.03	0.528 ± 0.003	0.583 ± 0.011
USAIR	0.88 ± 0.006	0.836 ± 0.01	0.557 ± 0.003	0.586 ± 0.012
YEAST	0.958 ± 0.002	0.919 ± 0.006	0.527 ± 0.002	0.549 ± 0.005
Average & std	0.912 ± 0.062	0.869 ± 0.066	0.547 ± 0.042	0.571 ± 0.037
ARXIV	0.9+-0.002	0.862+-0.005	0.53+-0.001	0.573+-0.001
BLOGCATALOG	0.849 ± 0.003	0.848 ± 0.003	0.585 ± 0.001	0.587 ± 0.001
CELEG	0.835 ± 0.006	0.807 ± 0.009	0.573 ± 0.006	0.588 ± 0.005
FACEBOOK	0.96 ± 0.002	0.958 ± 0.003	0.52 ± 0.001	0.521 ± 0.002
NS	0.966 ± 0.001	0.951 ± 0.012	0.492 ± 0.002	0.518 ± 0.006
PB	0.886 ± 0.003	0.882 ± 0.003	0.555 ± 0.001	0.555 ± 0.002
POWER	0.963 ± 0.003	0.835 ± 0.008	0.519 ± 0.001	0.585 ± 0.003
ROUTER	0.953 ± 0.003	0.829 ± 0.017	0.523 ± 0.002	0.584 ± 0.008
USAIR	0.895 ± 0.005	0.876 ± 0.019	0.547 ± 0.004	0.565 ± 0.014
YEAST	0.96 ± 0.002	0.917 ± 0.002	0.524 ± 0.001	0.55 ± 0.001
Average & std	0.919 ± 0.053	0.878 ± 0.054	0.538 ± 0.03	0.561 ± 0.028
ARXIV	0.953+-0.006	0.923+-0.007	0.525+-0.002	0.554+-0.002
BLOGCATALOG	0.737+-0.043	0.733+-0.044	0.744+-0.028	0.745+-0.028
CELEG	0.671+-0.01	0.645+-0.009	0.718+-0.036	0.726+-0.027
FACEBOOK	0.909+-0.016	0.906+-0.016	0.616+-0.034	0.617+-0.035
NS	0.966+-0.003	0.955+-0.008	0.511+-0.004	0.526+-0.003
PB	0.752+-0.037	0.742+-0.047	0.65+-0.028	0.653+-0.03
POWER	0.961+-0.003	0.831+-0.014	0.522+-0.001	0.588+-0.006
ROUTER	0.812+-0.024	0.59+-0.11	0.622+-0.017	0.708+-0.046
USAIR	0.785+-0.039	0.758+-0.039	0.639+-0.029	0.644+-0.026
YEAST	0.86+-0.012	0.833+-0.014	0.595+-0.004	0.604+-0.007
Average & std	0.828±0.103	0.777±0.117	0.624±0.078	0.646±0.071
ARXIV	0.954+-0.002	0.926+-0.002	0.529+-0.001	0.551+-0.001
BLOGCATALOG	0.741 ± 0.107	0.739 ± 0.108	0.682 ± 0.053	0.683 ± 0.053
CELEG	0.794 ± 0.007	0.75 ± 0.013	0.587 ± 0.004	0.609 ± 0.009
FACEBOOK	0.968 ± 0.001	0.966 ± 0.002	0.519 ± 0.001	0.52 ± 0.002
NS	0.973 ± 0.003	0.966 ± 0.008	0.497 ± 0.001	0.519 ± 0.007
PB	0.852 ± 0.002	0.84 ± 0.01	0.568 ± 0.002	0.573 ± 0.006
POWER	0.96 ± 0.001	0.839 ± 0.006	0.524 ± 0.002	0.582 ± 0.003
ROUTER	0.924 ± 0.009	0.741 ± 0.041	0.54 ± 0.004	0.621 ± 0.019
USAIR	0.853 ± 0.009	0.821 ± 0.015	0.581 ± 0.007	0.595 ± 0.013
YEAST	0.947 ± 0.004	0.904 ± 0.012	0.533 ± 0.002	0.556 ± 0.007
Average & std	0.89 ± 0.084	0.841 ± 0.09	0.559 ± 0.055	0.584 ± 0.051
ARXIV	0.978+-0.006	0.937+-0.004	0.553+-0.007	0.578+-0.003
BLOGCATALOG	0.879 ± 0.005	0.876 ± 0.004	0.584 ± 0.002	0.585 ± 0.002
CELEG	0.784 ± 0.012	0.756 ± 0.012	0.628 ± 0.004	0.63 ± 0.008
FACEBOOK	0.931 ± 0.018	0.924 ± 0.017	0.552 ± 0.007	0.554 ± 0.007
NS	0.957 ± 0.005	0.93 ± 0.015	0.531 ± 0.004	0.54 ± 0.012
PB	0.865 ± 0.003	0.863 ± 0.006	0.584 ± 0.002	0.585 ± 0.007
POWER	0.91 ± 0.014	0.814 ± 0.009	0.56 ± 0.001	0.591 ± 0.005
ROUTER	0.929 ± 0.011	0.799 ± 0.034	0.572 ± 0.007	0.614 ± 0.021
USAIR	0.904 ± 0.008	0.872 ± 0.021	0.567 ± 0.003	0.577 ± 0.012
YEAST	0.932 ± 0.007	0.89 ± 0.015	0.554 ± 0.003	0.565 ± 0.006
Average & std	0.899 ± 0.052	0.858 ± 0.058	0.57 ± 0.027	0.582 ± 0.028

Chapter 4

Conclusion and Future Work

This work aimed to explore the novel approaches of Geometrical Machine Learning for the problem of Link Prediction in graphs. Six state of the art Graph Neural Network models have been researched, trained and carefully analysed on a collection of various graph datasets. Moreover, these Graph Neural Network models were also compared to the traditional, heuristic approaches. Eventually, a novel approach yet unapplied to GNNs, of graph features called Graphlet Degree Signatures Milenković and Pržulj (2008) have been supplied to the models to observe a significantly positive impact on the learning process.

Except for few cases, the featureless Graph Neural Networks perform well, and especially highly on datasets with high node degree, achieving 0.82 Test AUC on average on the given ten datasets. Compared to the traditional approach of Link Prediction, the four chosen heuristics attain 0.806 Test AUC on average, which is almost identical performance to 0.819 of the selected featureless GNN models. The approaches however differ in their practicality, and scalability to larger datasets in which the GNNs provide a better alternative.

The supplied orbital features significantly increased the Test AUC, and improved learning process for most of the models, except GAT for which they even worsened the results. The supplied orbital features stabilised the learning procedure, as the loss and AUC curves became more stabilised and displayed less variability among the runs. The last remaining framework, SEAL outperforms all of the mentioned models at unwelcome computational cost.

In conclusion, both the featureless and without feature GNNs had proven to be a viable and successful alternative to the application of Link Prediction in static, undirected graphs, by attaining favourable, and reproducible results under acceptable time.

4.1 Future Work

4.1.1 Short term perspectives

1. Due to the vast number of datasets and models in this work, the hyper-parameters of the Graph Neural Networks were largely unoptimised. Therefore a hyper-parameter optimisation and subsequent analysis of the hyper-parameters could be undertaken to conclude the most impactful parameters of the models on the performance. This also predominantly increases the GNN performance on the datasets.

2. Another short-term perspective could be the optimisation of the SEAL K-hop subgraph sampler used in the SEAL section, which is used to generate the SEAL dataset object. It could be rewritten to handle multiple CPU cores and to speed up the entire process.

4.1.2 Long term perspectives

1. As the research on graph features generates new ideas every day, a further, more in depth study of different node features could be undertaken. This could easily be an idea for another MSc thesis, where a student researches features and feature generation approaches, and their performance impact on the GNN models.

2. This work explored Link Prediction on static, undirected and homogeneous graphs however another idea is to conduct research on different types of complex networks such as temporal, directed or heterogeneous ones.

Appendix A

Code

This Python codeblock shows how to utilise multiple CPU cores in order to compute the heuristic scores. In the case of FACEBOOK dataset, the observed speedup was 4 times (4 hours to 30 minutes) with 40 Intel Xeon Silver cores. Another option is to combine Numba and CUDA and move the process to GPU, however the CPU approach is more accessible due to the wider support of routines.

```
import multiprocessing as mp
import networkx as nx
from tqdm import tqdm
import numpy as np
import random as rand
from sklearn.metrics import roc_curve, roc_auc_score

def func(Graph,eb,return_list, e_subset):
    predictions = nx.heuristic_of_choice(Graph, e_subset)
    array = []
    for u,v,t in tqdm(predictions):
        array.append([u,v,t])
    list.append(array)

Graph = nx.read_gpickle('graph.gpickle')
frac = 0.1
e_subset = rand.sample(G.edges(), int(frac * G.number_of_edges()))
train = Graph.copy()

# Remove 10\% of the edges from train dataset for Test dataset
train.remove_edges_from(e_subset)

eb = nx.non_edges(train)
l = []
for u,v in tqdm(eb):
    l.append((u,v))
```

```

splits = np.array_split(np.array(l),number_of_splits)
manager = mp.Manager()
list = manager.list()
processes = []

for i in range(number_of_cores):
    p = mp.Process(target=func, args=(train,splits[i],list, e_subset))
    processes.append(p)
    p.start()

for process in processes:
    process.join()

#Predictions are now ready to be used
predictions = return_list._getvalue()
predictions = np.array(predictions)
preds = np.concatenate(predictions, axis=0)

#Next line should be parallelised the same way to reduce compute time.
score, label = zip(*[(s, (u,v) in e_subset) for (u,v,s) in preds])
fpr, tpr, _ = roc_curve(label, score)
auc = roc_auc_score(label, score)

plt.figure()
plt.plot(fpr,tpr)
```

Listing A.1: CPU Multicore Heuristic Computation

Bibliography

Robert Ackland et al. Mapping the us political blogosphere: Are conservative bloggers more prominent? In *BlogTalk Downunder 2005 Conference, Sydney*. BlogTalk Downunder 2005 Conference, Sydney, 2005.

Lada A Adamic and Eytan Adar. Friends and neighbors on the web. *Social networks*, 25(3):211–230, 2003.

Mohammad Al Hasan, Vineet Chaoji, Saeed Salem, and Mohammed Zaki. Link prediction using supervised learning. In *SDM06: workshop on link analysis, counter-terrorism and security*, volume 30, pages 798–805, 2006.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

V Batagelj and A Mrvar. Pajek datasets http://vlado. fmf. uni-lj. si/pub/networks/-data/mix. *USAir97. net*, 2006.

Weisfeiler Boris and A.A Lehman. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia*, 2(9):pp. 12–16, 1968.

Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pages 3844–3852, 2016.

Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734. IEEE, 2005.

Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

Weiwei Gu, Fei Gao, Xiaodan Lou, and Jiang Zhang. Link prediction via graph attention network. *arXiv*, pages arXiv–1910, 2019.

Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In Gael Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pages 1024–1034, 2017.

Charles R. Harris, K. Jarrod Millman, St'efan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fern'andez del R'ıo, Mark Wiebe, Pearu Peterson, Pierre G'erard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

Thomas Hocevar. Orca: Orbit counting algorithm, 2016.

J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

Paul Jaccard. Étude comparative de la distribution florale dans une portion des alpes et des jura. *Bull Soc Vaudoise Sci Nat*, 37:547–579, 1901.

Glen Jeh and Jennifer Widom. Simrank: a measure of structural-context similarity. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 538–543, 2002.

Leo Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1): 39–43, 1953.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016a.

Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016b.

Jon M Kleinberg. Authoritative sources in a hyperlinked environment. In *Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, pages 668–677, 1998.

Mark A Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal*, 37(2):233–243, 1991.

Ajay Kumar, Shashank Sheshar Singh, Kuldeep Singh, and Bhaskar Biswas. Link prediction techniques, applications, and performance: A survey. *Physica A: Statistical Mechanics and its Applications*, 553:124289, 2020.

Jure Leskovec and Andrej Krevl. Snap datasets: Stanford large network dataset collection, 2014.

Jure Leskovec and Rok Sosič. Snap: A general-purpose network analysis and graph-mining library. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8 (1):1, 2016.

David Liben-Nowell and Jon Kleinberg. The link-prediction problem for social networks. *Journal of the American society for information science and technology*, 58(7):1019–1031, 2004.

Weiyi Liu, Pin-Yu Chen, Hal Cooper, Min Hwan Oh, Sailung Yeung, and Toyotaro Suzumura. Can gan learn topological features of a graph? *arXiv preprint arXiv:1707.06197*, 2017.

Linyuan Lü and Tao Zhou. Link prediction in complex networks: A survey. *Physica A: statistical mechanics and its applications*, 390(6):1150–1170, 2011.

Aditya Krishna Menon and Charles Elkan. Link prediction via matrix factorization. In *Joint european conference on machine learning and knowledge discovery in databases*, pages 437–452. Springer, 2011.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

Tijana Milenković and Nataša Pržulj. Uncovering biological network function via graphlet degree signatures. *Cancer informatics*, 6:CIN–S680, 2008.

Mark EJ Newman. Clustering and preferential attachment in growing networks. *Physical review E*, 64(2):025102, 2001.

Mark EJ Newman. Finding community structure in networks using the eigenvectors of matrices. *Physical review E*, 74(3):036104, 2006.

The pandas development team. pandas-dev/pandas: Pandas, February 2020.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710, 2014.

Natasa Pržulj, Derek G Corneil, and Igor Jurisica. Modeling interactome: scale-free or geometric? *Bioinformatics*, 20(18):3508–3515, 2004.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

Neil Spring, Ratul Mahajan, David Wetherall, and Thomas Anderson. Measuring isp topologies with rocketfuel. *IEEE/ACM Transactions on networking*, 12(1):2–16, 2004.

V Strobel. Pold87/academic-keyword-occurrence: First release (version v1. 0.0). zenodo, 2018.

Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*, pages 1067–1077, 2015.

Lei Tang and Huan Liu. Leveraging social media networks for classification. *Data Mining and Knowledge Discovery*, 23(3):447–478, 2011.

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

Christian Von Mering, Roland Krause, Berend Snel, Michael Cornell, Stephen G Oliver, Stanley Fields, and Peer Bork. Comparative assessment of large-scale data sets of protein–protein interactions. *Nature*, 417(6887):399–403, 2002.

Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world'networks. *nature*, 393(6684):440–442, 1998.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

Jiaxuan You, Rex Ying, and Jure Leskovec. Position-aware graph neural networks. *arXiv preprint arXiv:1906.04817*, 2019.

Wayne W Zachary. An information flow model for conflict and fission in small groups. *Journal of anthropological research*, 33(4):452–473, 1977.

Reza Zafarani and Huan Liu. Social computing data repository at asu, 2009. *URL http://socialcomputing. asu. edu*, 2009.

Muhan Zhang and Yixin Chen. Weisfeiler-lehman neural machine for link prediction. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 575–583, 2017.

Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In *Advances in Neural Information Processing Systems*, pages 5165–5175, 2018.

Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Tao Zhou, Linyuan Lü, and Yi-Cheng Zhang. Predicting missing links via local information. *The European Physical Journal B*, 71(4):623–630, 2009.