The Name of the Title Is Hope

YUCHENG JIN* and G.K.M. TOBIN*, Fudan University, China
LARS THØRVÄLD, The Thørväld Group, Iceland
VALERIE BÉRANGER, Inria Paris-Rocquencourt, France
APARNA PATEL, Rajiv Gandhi University, India
HUIFEN CHAN, Tsinghua University, China
CHARLES PALMER, Palmer Research Laboratories, USA
JOHN SMITH, The Thørväld Group, Iceland
JULIUS P. KUMQUAT, The Kumquat Consortium, USA

A clear and well-documented LaTeX document is presented as an article formatted for publication by ACM in a conference proceedings or journal publication. Based on the "acmart" document class, this article presents and explains many of the common variations, as well as many of the formatting elements an author may use in the preparation of the documentation of their work.

CCS Concepts: • **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

Additional Key Words and Phrases: datasets, neural networks, gaze detection, text tagging

ACM Reference Format:
Yucheng Jin, G.K.M. Tobin, Lars Thørväld, Valerie Béranger, Aparna Patel, Huifen Chan, Charles Palmer, John Smith, and Julius P. Kumquat. 2018. The Name of the Title Is Hope. *J. ACM* 37, 4, Article 111 (August 2018), 3 pages. https://doi.org/XXXXXXX.XXXXXXX

1 PRELIMINARIES

Consider a graph $G = (V, E)$ with N nodes, where $V = v_1, v_2, \ldots, v_N$ denotes the set of nodes and $E \subset V \times V$ denotes the set of links. Let $A \in 0, 1^{N \times N}$ be the adjacency matrix, where $A_{i,j} = 1$ indicates that nodes v_i and v_j are connected, and 0 otherwise. Let $X \in \mathbb{R}^{N \times F}$ be the node feature matrix, where F is the number of raw node features and x_i represents the feature vector of node v_i (i.e., the i-th row of X). We use y to denote the label of each sample, which can be a node, edge, or graph depending on the task. We use a tilde symbol to denote the data generated by GDA methods, for instance, $\tilde{A}$ denotes the augmented adjacency matrix, $\tilde{x}_i$ denotes the augmented feature vector of node v_i, and so forth.

*Both authors contributed equally to this research.

Authors' addresses: Yucheng Jin, trovato@corporation.com; G.K.M. Tobin, webmaster@marysville-ohio.com, Fudan University, P.O. Box 1212, Shanghai, Shanghai, China, 43017-6221; Lars Thørväld, The Thørväld Group, 1 Thørväld Circle, Hekla, Iceland, larst@affiliation.org; Valerie Béranger, Inria Paris-Rocquencourt, Rocquencourt, France; Aparna Patel, Rajiv Gandhi University, Rono-Hills, Doimukh, Arunachal Pradesh, India; Huifen Chan, Tsinghua University, 30 Shuangqing Rd, Haidian Qu, Beijing Shi, China; Charles Palmer, Palmer Research Laboratories, 8600 Datapoint Drive, San Antonio, Texas, USA, 78229, cpalmer@prl.com; John Smith, The Thørväld Group, 1 Thørväld Circle, Hekla, Iceland, jsmith@affiliation.org; Julius P. Kumquat, The Kumquat Consortium, New York, USA, jpkumquat@consortium.net.

0004-5411/2018/8-ART111 $15.00
https://doi.org/XXXXXXX.XXXXXXX

Data augmentation (DA) refers to a set of techniques used to increase or generate training data without the need to collect or label more data. Common DA techniques involve creating slightly modified copies of existing data or generating synthetic data based on existing data. By doing so, the augmented data can help prevent overfitting in data-driven models [**85**]. Commonly used augmentation operations in machine learning, computer vision (CV) [**15**], and natural language processing (NLP) [**26**] include cropping, flipping, and back-translation. In contrast, geometric machine learning (GML) deals with non-Euclidean and irregular data, often represented by graphs. Unlike regular data, such as grids (e.g., images) and sequences (e.g., sentences), structured augmentation operations commonly used in CV and NLP cannot be easily applied to graph data. In node-level and edge-level tasks, objects are interconnected and non-i.i.d., which means that GDA techniques typically modify the entire dataset (graph) instead of a specific data object (nodes or edges) in isolation. A GDA method can generally be defined as a transformation function $f : G \rightarrow \tilde{G}$. The transformation function f can be either rule-based or learnable, and the augmented graph $\tilde{G}$ contains the augmented adjacency matrix $\tilde{A}$ and node feature matrix $\tilde{X}$ (and optionally augmented edge features, node or graph labels). The augmentation function f is not necessarily deterministic, meaning that the same f may generate multiple versions of the augmented graph $\tilde{G}$, and the model may use one or multiple of these augmentations as required for training.

1.1 Graph Adversarial Training

Adversarial training is a widely used defense mechanism against adversarial attacks in computer vision [**31**], and has also been extended to the graph domain [**10**; **17**; **18**; **20**; **25**; **41**; **57**]. Unlike graph structure learning, graph adversarial training does not aim to find an optimal graph structure. Instead, it augments input graphs with adversarial patterns during model training by perturbing node features or graph structure. The resulting adversarially trained models are expected to tolerate adversarial perturbations in graph data, leading to better generalization and robustness performance at test time. At the core of adversarial training is the injection of adversarial examples into the training set, which enables the trained model to properly predict test adversarial examples. Therefore, we can use this strategy to enhance the robustness of GNNs as follows:

$$\min_{\theta} \max_{\Delta_A \in P_A \Delta_X \in P_X} L_{train}(g_\theta(A + \Delta_A, X + \Delta_X)) \tag{1}$$

where L_{train} denotes the training loss for the downstream task; Δ_A and Δ_X stand for the perturbation on A, X,respectively; P_A and P_X denote the perturbation space. From the bi-level optimization problem in Equation (11), we can observe that adversarial training generates perturbations that maximize the prediction loss and updates model parameters to minimize the prediction loss. The process of generating perturbations (i.e., $A + \Delta_A, X + \Delta_X$) can be viewed as adversarial data augmentation and we can leverage such augmentations to improve the model robustness and generalization.

Dai et al. [**17**] proposed a simple adversarial training strategy for augmenting the adjacency matrix by randomly dropping edges during training, without any optimization on the graph data. Although this approach does not provide significant improvement, it still shows some benefits in enhancing the robustness of GNNs. This finding is consistent with that of Z ̈ugner and G ̈unnemann [**146**]. On the other hand, Xu et al. [**116**] utilized projected gradient descent (PGD) to optimize the bi-level problem and generate perturbations on the discrete structure, achieving a significant improvement in robust performance. Similarly, Chen et al. [**9**] and Dai et al. [**18**] used existing adversarial attacks to modify the input graph structure during adversarial training for network embedding methods. Additionally, Suresh et al. [**92**] proposed to learn to drop edges during graph augmentation to capture the minimal information needed to classify each graph.

In contrast to augmenting the adjacency matrix, some works focus on perturbing the input features as adversarial examples. For example, Feng et al. [**25**] proposed dynamic regularization to reconstruct graph smoothness and constrain the divergence between the prediction of the target node and its connected nodes. Deng et al. [**20**] introduced batch virtual adversarial training to promote the smoothness of GNNs and improve their resilience against adversarial perturbations. Kong et al. [**57**] developed FLAG, which utilizes adversarial training to iteratively augment the node features with gradient-based adversarial perturbations, and enhances the performance of GNNs in node classification, link prediction, and graph classification tasks. Moreover, Z ̈ugner and G ̈unnemann [**145**] investigated the certifiable robustness of GNNs with respect to perturbations of node attributes and proposed a robust training scheme inspired by the certificates. Several other variants of adversarial training on perturbing node features are introduced in [**41**; **103**].

1.2 Rationalization

Rationalization in machine learning refers to the process of explaining model predictions by identifying the input features that are most relevant to the prediction. In the graph domain, rationalization is typically implemented as a form of augmented graph data that includes intrinsically learned subgraphs representing key features of the graph. These subgraphs are used to inform model decisions and provide explanations for the model's predictions. Rationalization is commonly applied to graph property prediction or graph classification tasks, such as drug discovery, molecular and polymer datasets, etc.

Early rationalization methods in the graph domain focused on improving interpretability and general graph classification performance. Yu et al. [**121**] proposed the Graph Information bottleneck framework (GIB) that learns to generate maximally informative and compressed subgraphs by leveraging a bi-level optimization scheme and a novel connectivity loss. Miao et al. [**73**] proposed GSAT to better learn and select task-relevant subgraphs by injecting stochasticity into the attention weights to constrain information from task-irrelevant components. Liu et al. [**64**] proposed GREA, a rationalization method that improves rationale identification by separating the rationale and environment and subsequently replacing the environment to generate augmented virtual data examples.

Rationalization models also help solve data bias and out-of-distribution (OOD) problems for graph property prediction tasks, providing better interpretation and generalization. Wu et al. [**111**] proposed DIR to generate distribution perturbation on training data with causal intervention. They created a rationale generator that separates causal and non-causal graphs, applies causal intervention to create perturbed distributions, and then jointly learn both the causal and non-causal representation to minimize invariant risk. Similarly, Chen et al. [**12**] proposed CIGA to model the graph generation process and the interactions between invariant and spurious features with Structural Causal Models (SCM). The resulting subgraphs generated by CIGA maximally preserves the invariant intra-class information. Li et al. [**62**] proposed GIL, a GNN-based subgraph generator that identifies potentially invariant subgraphs, infers latent environment labels for variant subgraphs, and jointly optimizes all modules.

In summary, rationalization methods in the graph domain are intrinsically interpretable models that include a rationalization component in the model, as opposed to post-hoc explanation methods. These methods help improve interpretability, generalization, and solve data bias and OOD problems for graph property prediction tasks.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009