EXPLAINING THE BOX: A GLIMPSE INSIDE DECISIONS OF MODERN AI

By

MINH NHAT VU

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

2023

”But the problem, you see, when you ask why something happens, how does a person answer why something happens? For example, Aunt Minnie is in the hospital. Why? Because she went out, slipped on the ice, and broke her hip. That satisfies people. It satisfies, but it wouldn’t satisfy someone who came from another planet and knew nothing about why when you break your hip do you go to the hospital. How do you get to the hospital when the hip is broken? Well, because her husband, seeing that her hip was broken, called the hospital up and sent somebody to get her. All that is understood by people. And when you explain a why, you have to be in some framework that you allow something to be true. Otherwise, you’re perpetually asking why.” - Richard Feynman

The quest of explaining any decision involves the question *why*, which can endlessly lead to a series of further *why*-s. How can we be sure that we have successfully explained anything, especially when it comes to the decisions of complex modern AI?

I’ve concluded that there are times when we simply need to believe. However, if we ultimately need to believe in something, why can’t we just simply believe the things we wish to explain and save us from all the nuisance? Oh, I simply believe that endlessly following the *why*-s is the reason for my existence.

For that, this work is dedicated to all those who continuously ask *Why*.

ACKNOWLEDGEMENTS

I would like to thank Dad and Mom. Besides their unwavering love and support, I am forever grateful that they always encourage me toward morality, honesty, and kindness. Without their love, and guidance, I would never become who I am.

I want to express my gratitude to my advisor, Dr. Thai, for continuously providing me with excellent research opportunities, inspiration, and belief. Her great knowledge, enthusiasm, and dedication to scientific research have been instrumental in shaping my professional identity.

I appreciate for the knowledge, insights, and encouragement of my friends and colleagues: Lan, Truc, Raed, and Huy Mai. It has been an honor and a privilege to work with such talented and kind people.

I want to thank Kim, my sweet little sister. She always supports me during the time of this research. She is a miracle.

Finally, I would like to thank my committee members, Dr. Kevin Butler, Dr. Kejun Huang, and Dr. Sandip Ray, for their support throughout my academic journey at UF. Their teaching, guidance, and advice have helped me a lot in my Ph.D. in UF.

TABLE OF CONTENTS

page

ACKNOWLEDGEMENTS 4

LIST OF TABLES 9

LIST OF FIGURES 13

LIST OF OBJECTS 14

ABSTRACT 15

CHAPTER

1 INTRODUCTION 17

1.1 Explainable AI and Major Approaches 17
1.2 Properties of Explainer and Explanations 20
1.3 Overview of Some Local Explanation Methods 22
1.4 Dissertation Outline 24

2 C-EVAL: A UNIFIED METRIC TO EVALUATE FEATURE-BASED EXPLANATIONS VIA PERTURBATION 28

2.1 C-Eval of Explanation 30
2.2 Computing C-Eval 33
2.3 C-Eval and the Importance of Features 36
2.3.1 C-Eval in Affine Classifiers 37
2.3.2 C-Eval in General Non-Affine Classifiers. 38
2.4 Beyond C-Eval 39
2.4.1 C-Eval Plot 39
2.4.2 C-Eval on Adversarial-Trained Models 40
2.5 Experimental Results 42
2.5.1 System Specifications and Source Code 42
2.5.2 Simulations on MNIST Dataset 42
2.5.3 Simulations on Caltech101 Dataset 44
2.5.4 Similarity of C-Eval and Log-Odds Functions in MNIST 45
2.5.5 Overall Evaluations of Explanations Using C-Eval 45
2.6 Conclusion 46

3 EMAP: EXPLAINABLE AI WITH MANIFOLD-BASED PERTURBATIONS 53

3.1 Related Work 55
3.2 Preliminaries 56
3.2.1 Explainers 56
3.2.2 Topological-Awareness Perturbations for Explaining Tasks 57
3.2.3 Attack Framework 58
3.3 Analysis of Discrete Gromov-Hausdorff Distances of Perturbations 59
3.4 Persistent Homology Analysis With the Bottleneck Distance 63

3.5 EMaP Algorithm 65
3.5.1 Algorithm Overview 67
3.5.2 The Mapper 67
3.5.3 EMaP Orthogonal Perturbations 67
3.5.4 Local-Subspace Approximation 68
3.6 Experiments 70
3.6.1 Experimental Settings 70
3.6.2 Explaining Performance of Explanation Methods 72
3.6.3 Perturbation's Robustness 75
3.6.4 Computational Resource and Run Time 75
3.7 Conclusion, Limitations and Future Research 76

4 NEUCEPT: LEARN NEURAL NETWORKS' MECHANISM VIA CRITICAL NEURONS WITH PRECISION GUARANTEE 83

4.1 Critical Neurons Identification with NeuCEPT-Discovery 84
4.1.1 Problem Formulation 85
4.1.2 Solutions with Precision Guarantee 86
4.2 Information-Theoretic Interpretation and NeuCEPT-Learning of Critical Neurons . . . 90
4.2.1 Information-Theoretic Interpretation 90
4.2.2 NeuCEPT-Learning on Critical Neurons 91
4.3 Experiments: Setting and Results 93
4.3.1 Prior-Knowledge Training (PKT) 93
4.3.2 Experimental Setting 94
4.3.3 Explainability Power 95
4.3.4 Non-Redundancy. 96
4.3.5 Running-Time. 97
4.4 Case Studies 97
4.5 Conclusion 99

5 PGM-EXPLAINER: PROBABILISTIC GRAPHICAL MODEL EXPLANATIONS FOR GRAPH NEURAL NETWORKS 106

5.1 Preliminaries 108
5.2 PGM-Explainer: Probabilistic Graphical Model Explanations for GNNs 110
5.2.1 Data Generation 111
5.2.2 Variables Selections 112
5.2.3 Structure Learning 114
5.3 Experiments 118
5.3.1 Datasets and Experiment Settings 118
5.3.2 Results on Synthetic Datasets 120
5.3.3 Results on Real-world Datasets 121
5.4 Conclusion 122

6 UNEXPLAINABLE AI: ON THE LIMIT OF EXPLAINING BLACK-BOX TEMPORAL GRAPH NEURAL NETWORKS 127

6.1 Related Works 128
6.2 Preliminaries 129
6.3 Unidentifiable Proof for Neural Networks 131
6.3.1 The Interpretable Domain 131
6.3.2 The Transparent Model 132
6.3.3 The Unidentifiable Proof 133
6.4 Unidentifiable Proof for Node-Perturbation 133
6.4.1 The Training Task 134
6.4.2 The Models 134
6.4.3 The Forwarding Computation of Φ_1^v and Φ_2^v 136
6.4.4 The Transparent Models of Φ_1^v and Φ_2^v 136
6.4.5 Unidentifiable Proof 138
6.5 Unidentifiable Proof for Edge-Perturbation 139
6.5.1 The Training Task and the Models 139
6.5.2 The Transparent Models and Unidentifiable Proof 141
6.6 Unidentifiable Proof for Node-and-Edge Perturbation in TGNN 141
6.6.1 The Training Task 142
6.6.2 The Models 142
6.6.3 Forwarding computation of Φ_2^a 143
6.6.4 The Transparent Models of Φ_1^a and Φ_2^a 143
6.6.5 Unidentifiable Proof 144
6.7 Experiments 145
6.7.1 Synthetic Experiment on Node-perturbation 146
6.7.2 Experiments on Node-and-Edge-perturbation 146
6.8 Discussion and Conclusion 147

APPENDIX

A EXPERIMENTS OF C-EVAL ON CIFAR10 154

B EXAMPLES OF EXPLANATIONS OF MNIST, CIFAR10 AND CALTECH101 155

C VISUALIZATIONS OF SYNTHETIC DATA AND EXPLANATIONS GENERATED WITH EMAP 158

D ADDITIVE FEATURE ATTRIBUTION METHODS UNIFY EXISTING EXPLAINERS FOR GNNS 163

E EXAMPLE OF PGM-EXPLAINER WITH AND WITHOUT *NO-CHILD* CONSTRAINT 165

F PROBABILISTIC GRAPHICAL MODEL PRELIMINARIES 167
F.1 I-Map and Perfect Map 167
F.2 Markov-Blanket 168
F.3 Structure Learning and Bayesian Information Criterion (BIC) Score 168

G BAYESIAN NETWORK AND DYNAMIC BAYESIAN NETWORK 170
LIST OF REFERENCES 172
BIOGRAPHICAL SKETCH 184

LIST OF TABLES

Tables page

1-1 Scope of works 24

3-1 The parameters of the Bottleneck distance's experiments on the synthetic datasets. 78

3-2 The parameters of the Bottleneck distance's experiments on the real-world datasets. 78

3-3 The normalized H_0 and H_1 Bottleneck distances for Gaussian, projection, and orthogonal perturbations on synthetic datasets. 78

3-4 Run time of EMaP and LIME in generating explanations using 1000 perturbations per explanation. 78

4-1 The running time in seconds of explaining methods. 104

5-1 Parameters of synthetic datasets. 124

5-2 Models' accuracy and number of sampled data used by PGM-Explainer 124

5-3 Accuracy of Explainers on Synthetic Datasets. 125

5-4 Precision of Explainers on Trust Signed Networks datasets. 125

6-1 Summary of perturbation methods used by explainers and the scope of our results. 150

6-2 The constant features of the TGNNs in G_v's proof. 151

6-3 The constant features of the TGNNs in G_a's proof. 152

6-4 Hidden features of the TGNN Φ_2^a 152

6-5 GNNExplainer fails to detect the ground-truth explanations of the synthetic task 152

LIST OF FIGURES

Figures page

2-1 Explanations generated by three feature-based local explainers for the prediction *Pembroke* of Inception-v3 model. 47

2-2 Example comparing the importance of features included in an explanation and features in a dummy mask using c-Eval. 47

2-3 Distortions between perturbations and the original images. 47

2-4 Scatter plot of c-Eval computed by gradient-based attacks vs c-Eval computed by CW attack. 48

2-5 c-Eval in 2D affine classifier. 48

2-6 c-Eval in non-linear classifier. 48

2-7 Example of nearly affine instance on Inception-v3. 49

2-8 Demonstration of the condition $c_{f,\boldsymbol{x}}(\emptyset) \approx \hat{c}_{f,\boldsymbol{x}}(\emptyset)$ in VGG19 and ResNet50 49

2-9 c-Eval plot of LIME with different sample rates 50

2-10 Correlation between c-Eval on non-robust and adversarial-trained models. 50

2-11 The distributions and the averages of c-Eval of 8 explainers 51

2-12 Distributions of c-Eval computed by GSA and IGA for four explainers in Caltech101 dataset. 51

2-13 c-Eval for 6 explainers on 1000 images of MNIST. 52

3-1 Visualization of perturbations with the same magnitude generated from a point cloud of a 2-dimensional spiral. 76

3-2 Overview of EMaP's perturbation 76

3-3 Example showing how leveraging topological information can be beneficial for the explanation task. 77

3-4 A 1-D example showing the impact of perturbation's direction. 77

3-5 The discriminator-based attack framework 77

3-6 The normalized H_0 and H_1 Bottleneck distances for orthogonal and projection perturbations on 3 real-world datasets at different noise levels. 79

3-7 The normalized H_0 and H_1 Bottleneck distances for orthogonal and projection perturbations on 2 image datasets at different noise levels. 79

3-8 Euclidean distances computing in the input space vs. that on manifold space 79

3-9 Examples of explanations of different methods in MNIST and Fashion-MNIST. 80

3-10 The precision and recall of explanations returned by Greedy, LIME, Parzen, and LIME-EMaP 80

3-11 RDT fidelity scores of different perturbation-based methods on MNIST and Fashion-MNIST 81

3-12 Infidelity scores of different perturbation-based methods on MNIST and Fashion-MNIST 81

3-13 Log-odds scores of different perturbation-based methods on MNIST and Fashion-MNIST 81

3-14 True-Positive and True-Negative rates of the discriminators on perturbations of different methods on MNIST dataset 82

3-15 True-Positive and True-Negative rates of the discriminators on perturbations of different methods on Fashion-MNIST dataset 82

3-16 True-Positive and True-Negative rates of the discriminators on perturbations of different methods on Communities and Crime dataset 82

3-17 True-Positive and True-Negative rates of the discriminators on perturbations of different methods on German Credit dataset. 82

4-1 Are the mechanisms behind the two predictions of DNNs the same? 100

4-2 Failures of feature-attribution explanations. 100

4-3 Overall architecture of NeuCEPT 101

4-4 The probabilistic intuition of the neurons’ activation and mechanism 101

4-5 Examples of NeuCEPT’s outputs on Inception-v3. 101

4-6 Prior-knowledge training 102

4-7 CE (bits) resulted from K-mean (KM) and Agglomerative clustering (AC) clustering on activation of neurons selected by different methods. 102

4-8 Ablation tests on LeNet. The x-axis and y-axis are the noise levels and the test accuracy. 102

4-9 Ablation tests on VGG. The x-axis and y-axis are the noise levels and the test accuracy, respectively. 102

4-10 The CE differences (bits) between the conventionally trained model and the PKT model. 103

4-11 Ablation results at the input layer of LeNet. 103

4-12 Using LP with NeuCEPT can help detect the linear separability of the two models 104

4-13 Results obtained by applying NeuCEPT with $K = 4$ on classes *goldfish* and *bee* at the *Mixed6e* block of Inception-v3. 105

4-14 NeuCEPT discovers a subclass of unreliable predictions with precision of 36.7% (compared to 60.6% on the whole class) in CheXNet. 105

5-1 Example of PGM-explanation of a GNN's prediction. 123

5-2 The architecture of PGM-Explainer. 123

5-3 Examples and distributions of scores of explanations generated by different explainers on the MNIST SuperPixel-Graph dataset. 124

5-4 The distribution of scores given by end-users on the explanations 124

6-1 Tasks for different classes of perturbation-based explanation methods. 150

6-2 An example of a DBN and its unroll BN.. 150

6-3 The association among variables of the explanation DBN, the input nodes and the messages in the TGNN 151

6-4 Necessary conditions for explanations under the Existence assumption. 151

6-5 The general TGNN model. 151

6-6 The DBNs explaining the two T-GNNs Φ_1^v and Φ_2^v. 151

6-7 The DBNs for Unidentifiable Proof of $\mathcal{G}_e$. 151

6-8 The DBNs for Unidentifiable Proof of $\mathcal{G}_a$. 151

6-9 The synthetic static task demonstrating the failure of node-perturbation explanation methods. 152

6-10 The synthetic temporal task demonstrating the failure of node-and-edge perturbation-based explanation methods. 152

6-11 The temporal activations of TGNNs in synthetic temporal task demonstrating the failure of Node-and-edge perturbation-based explanation methods. 153

6-12 The real-world temporal messages demonstrating the failure of Node-and-edge perturbation-based explanation methods. 153

6-13 The components for Node-perturbation Unidentifiable Proof of GNNs. 153

6-14 The BNs for Edge-perturbation Unidentifiable Proof of GNNs. 153

A-1 Distributions of *c*-Eval on CIFAR10 dataset. 154

B-1 Some examples of explanations and *c*-Eval on MNIST. 155

B-2 Some examples of Explanations and *c*-Eval on CIFAR10. 156

B-3 Some examples of Explanations and *c*-Eval on Caltech101. 157

C-1 Visualizations of some synthetic data in experiments on the normalized H_0 and H_1 Bottleneck distances. 158
C-2 Visualization of EMaP-LIME explanations of the Multi-polarity-books review 159
C-3 Visualization of EMaP-LIME explanations of the Multi-polarity-kitchen review 159
C-4 Visualization of EMaP-LIME explanations of the MNIST dataset. 160
C-5 Examples of explanations in MNIST. 161
C-6 Examples of explanations in Fashion-MNIST 162
E-1 Example of PGM-explanation with and without *no-child* constraint. 165
G-1 Example of the 2TBN and its unroll BN. 171

LIST OF ALGORITHMS

Algorithms page

3-1 EMaP algorithm . . . 66

3-2 gen_perturbation . . . 68

3-3 get_local_subspace . . . 69

4-1 NeuCEPT-discovery. . . . 87

4-2 NeuCEPT-learning. . . . 92

5-1 Generating PGM explanation for GNN without no-child constraint . . . 125

5-2 Generating PGM explanation for GNN with no-child constraint . . . 126

.

Abstract of Dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

EXPLAINING THE BOX: A GLIMPSE INSIDE DECISIONS OF MODERN AI

By

Minh Nhat Vu

December 2023

Chair: My T. Thai
Major: Computer Science

Recent years have observed a swift adoption of modern Artificial Intelligence (AI) models in real-world applications, especially those in critical and sensitive domains such as health care, public policy, and autonomous systems. Ensuring the safety, security, and reliability of AI systems has become a paramount concern. In response, this dissertation presents a collection of heuristic and theoretical research results shedding light on the decision-making processes of contemporary AI systems.

First, we introduce the c-Eval metric and the multiple algorithms to evaluate the faithfulness of local explanation. Given a prediction of a deep neural network and its explanation, c-Eval is the minimum-distortion perturbation that successfully alters the prediction while keeping the explanatory features unchanged.

Second, we introduce a novel perturbation scheme so that more faithful and robust explanations of perturbation-based explanation methods can be obtained. In particular, the study focuses on the impact of perturbing directions on the data topology. We show that perturbing along the orthogonal directions of the input manifold better preserves the data topology, both in the worst-case analysis of the discrete Gromov-Hausdorff distance and in the average-case analysis via persistent homology.

To obtain more in-depth explanations for neural networks, we propose NeuCEPT, a method to identify critical neurons that are important to the model's local predictions and learn their

underlying mechanisms. We also provide a theoretical analysis to efficiently solve for critical neurons while keeping the precision under control.

While many explanation methods have been developed for deep models operating on grid-like data, e.g. time series, text, and images, the counterparts for graph data are lacking. In response, we introduce PGM-Explainer, a Probabilistic Graphical Model (PGM) explainer for Graph Neural Networks (GNNs). We theoretically show that the resulting explanation guarantees to include all statistical information regarding the target of the explanation.

Finally, to circumvent the lack of internal information on the explained models, black-box explainers rely on the responses of the model on some perturbations of input data. We theoretically point out that this lack of internal access limits perturbation-based methods from uncovering certain crucial information about the predictions generated by Temporal Graph Neural Networks (TGNNs), i.e., a class of modern AI models.

The results of the works in this manuscript have a strong impact on a wide range of AI-related applications, especially those that require high levels of trust, safety, and security.

CHAPTER 1
INTRODUCTION

The success of neural networks has resulted in the widespread adoption of AI and machine learning (ML) in many real-world applications [14, 49]. However, most currently deployed AI systems are still limited by the inability to explain their decisions and predictions to humans [96, 8]. This lack of transparency is significantly hindering our understanding, trust, control, and deployment of AI systems [107, 53, 134]. In 2019, the Defense Advanced Research Projects Agency (DARPA) introduced the term *Explainable AI* (XAI) [52] and it has been receiving a lot of popularity ever since. While that was not the first time computer scientists investigated the question of why behind ML [103], the introduction of the term aligned with a sharp increase in XAI-related researches [8]. The surveys [19, 32] are highly recommended for a more detailed discussion of this topic.

This chapter provides a brief literature review on XAI (Section 1.1), some properties of explainers/explanations (Section 1.2, and the overview of some popular local explanation methods 1.3). Finally, we conclude this chapter with the outline of this dissertation (Section 1.4).

1.1 Explainable AI and Major Approaches

Despite a large number of research, the exact definition of XAI is not yet consolidated. For the context of this manuscript, XAI refers to *set of processes and methods that allows human users to understand the outputs generated by machine learning algorithms*. Since XAI covers many methods, techniques, and research, it would be beneficial to first categorize them into smaller classes of methods. Particularly, XAI methods and techniques can be categorized based on different criteria. We describe them in the remainder of this section.

Global vs. Local: Global methods focus on the behavior of ML on a macro level. For example, they are interested in understanding how the model generally makes decisions, what weights or features determine some major classes of predictions (in the case of classification models), or how a particular group of features are transformed during the model's decision-making process. Some well-known global methods are Partial Dependence Plot (PDP) [43], Accumulated Local Effects (ALE) Plot [7], Permutation Feature Importance [41] and

Global Surrogate [59, 77, 151]. These methods either examine the impact or contribution of a single feature or a group of features on the model's decision or approximate the explained model with a simpler model for interpretation. Local methods, on the other hand, explain individual predictions. The term local not only refers to the target of explanation (individual prediction) but also the focus of explanation methods, which are the behaviors of the model in the vicinity of the input data. This kind of information that local methods provide is, in general, more relevant and important to end-users. For example, they aim to answer why the model returns some predictions on some given inputs or how much some features contribute to a specific prediction. Notable local methods are LIME [111], SHAP [86], KernelSHAP [129], DeepLIFT [122], Integrated-gradients [131], Grad-CAM [121], CXPlain [120], and many others [159, 21, 155, 140, 127].

Pre-Model vs. In-Model vs. Post-Model: XAI methods can be classified based on when the methods are applicable [13]. Pre-model methods focus on the data rather than the machine learning model. Their goals are to obtain meaningful intuitive properties that help to achieve explainability/interpretability in the data, especially for those with high dimensions or complex structures. Thus, these methods are closely related to data analysis techniques such as Principal Component Analysis (PCA) [65] and t-SNE [136], UMAP [89], and clustering methods. The focus of in-model methods is to measure and/or improve the interpretability of the ML models. This objective is normally obtained by imposing additional constraints on the models. On the other hand, post-model methods aim to explain the outputs of the ML models. These two classes will be discussed in more detail in the following paragraphs about intrinsic and post-hoc methods.

Intrinsic vs. Post-Hoc: While intrinsic methods generally aim to design more interpretable models, post-hoc methods focus on analyzing the models' predictions to achieve explainability [83]. Another interesting viewpoint on intrinsic vs. post-hoc is that the former answers the question of *how the model works* while the latter answers *what else can the model tell us* [83]. To obtain more interpretable models, intrinsic methods impose multiple desirable properties on the models' parameters and hidden representations such as sparsity, monotonicity, causality, and constraints from domain knowledge [115]. Post-hoc techniques, on the other hand, are applied to models after their training. This class of methods covers a large portion of XAI research since they are normally model-agnostic (independent of the explained model) and be used in a wide range of applications. Well-known Post-hoc methods are LIME [111], SHAP [86], DeepLIFT [122], and some others [120, 155, 140].

Model-Specific vs. Model-Agnostic: In many practical contexts, we have prior knowledge on the explained models, e.g. architectures, types of layers, and/or neurons' coupling. While XAI methods can leverage that knowledge for better explanations, it might prevent them from being applicable to a wider range of models. Those methods fall into the class of model-specific methods [122, 131, 127, 121]. On the contrary, methods working regardless of the model's architectures are known as model-agnostic[111, 86, 120, 140, 155]. It is noteworthy to point out that the model-agnostic methods can be used in combination with model-specific methods to improve confidence and sanity check on both the model's predictions and the quality of the explanations [122].

Explanatory domain: The output of XAI methods can be of different forms. In this manuscript, the term *explanatory domain* is used to refer to the domain of the representations of the explanations, which is normally a vector space. Different explanatory domains can provide users with different types of information. The most common choice is scoring on the input features. In particular, many XAI methods return a set of features or some summary statistics for the features such as importance weights, attribution scores, or feature sensitivities. These methods are also known as *feature-based explanation methods*. Depending on the input's domain, the explanations can have some specific forms such as heat-maps for predictions of images [121, 111] or low-dimensional layout to visualize activations in a trained network [143]. On the other hand, intrinsic methods return internal information about the explained model. For example, weights in linear models, critical neurons, and critical paths in deep neural networks [143, 157]. Prototype-explanation methods give input data as explanations. They are also known as example-based explanation methods, which are more common for text and image data. Popular methods of this class are the Counterfactual Explanations [144], the Prototypes and Criticisms [68], and the Influence Functions [72]. Another common choice of interpretable domain is the surrogate of interpretable models. Specifically, the local behavior of a complex ML model might be explained by a more simple interpretable model approximating it. This class of methods is commonly called surrogate-methods [111, 59, 77, 151].

1.2 Properties of Explainer and Explanations

To compare and assess explainers and their explanations, Robnik-Sikonja et al. [113] introduce some of the desirable properties of explainers and explanations. In this section, we briefly describe those notions. It is noteworthy to point out beforehand that, it remains unclear how to accurately measure and determine their relevance to particular use cases. Therefore, a significant challenge is to establish a formalized method for calculating these properties. Some key properties of explainers are:

1. *Expressive power*: this is the explanatory domain of the explanations. Some examples are propositional logic, first-order logic, finite state machines, histograms, decision trees, and linear models. The choice of explanatory domain not only determines the type of information that the explainer can convey, e.g., linear relationship in linear models but also limits the amount of information that the explainer can deliver. For instance, if the explained model operates in a non-linear manner, explanations in the form of linear models can not point out that information to users.

2. *Translucency*: this notion describes how much the explainer looks inside the explained model to generate the explanations. By this definition, model-specific methods are highly translucent while model-agnostic methods are non-translucent. Typically, model-specific methods normally look at the model's activation [121], parameters, and gradients [131].

3. *Portability*: portability describes the range of models/applications that the explanation methods can be deployed. It then follows that model-agnostic methods are highly portable while model-specific methods have low portability. Note that portability does not only

depend on the model's architecture but also on the availability of information on the model during the explanation process.

4. *Algorithmic complexity*: this measures the computational complexity of the explanation methods. Algorithmic complexity is generally more important for applications that require short reaction time such as streaming or autonomous systems.

Robnik-Sikonja et al. [113] also point out some important notions of explanations returned by explanation methods:

1. *Accuracy*: this refers to the general ability of explanations, which assesses the extent to which an explanation for a particular prediction can be generalized to other predictions that have not been observed. For example, if the explanation takes the form of rules, it is important to determine whether these rules are broad enough to cover new, unseen instances.

2. *Faithfulness/Fidelity*: (local) faithfulness/fidelity captures the correctness of explanations in capturing and describing the behavior of the explained model in the proximity of the explaining predictions. It is important to note that local fidelity does not imply global fidelity. Particularly, features that are important in a local context may not necessarily be important in the broader context of the model.

3. *Consistency*: this measures the similarity among explanations generated on different models trained on the same setting. It remains controversial whether high consistency is beneficial [93] since the same prediction might be reasonably generated from different features [16].

4. *Stability*: stability measures the similarity of explanations for similar inputs of the same models. This can be considered as the instance-wise counterpart of consistency. Stability suffers the same issue as consistency since similar predictions of similar inputs might be the results of different features and methodologies.

5. *Comprehensibility*: this captures how well humans can understand the explanations. While this notion is important, it is challenging to quantify comprehensibility since it depends heavily on the audience and the context [19]. The explanatory domain also has a significant influence on comprehensibility since a transformed [93].

6. *Certainty*: explanation's certainty measures the confidence of the model on its predictions. In some recently introduced neural networks, in addition to the predictions, the model can return some evidence about how much it is confident about its decisions. Some examples of certainty measures are the True-Class-Probability [23] and the Trust Score [63].

7. *Representativeness*: it is the scope of the explanation, e.g., does the explanation cover the overall behavior of the explained model or just a part of it.

1.3 Overview of Some Local Explanation Methods

This section provides an overview of some local explanation methods examined in this dissertation. We use the standard setting of the learning tasks where the set of input X is sampled from a distribution on $\mathbb{R}^N$. We address the explained model by its forwarding f mapping each input's space to a prediction y. An explanation of prediction $y = f(x)$ can be obtained by running an explainer g on x and f. We denote such an explanation by $g(x)$. The range of $g(x)$ is the explanatory domain.

1. *Gradient Explanation* (Grad) [123] uses $g_{grad}(x) = \partial f(x)/\partial x$ to explain the prediction of the model on the input x. It captures the change in the $f(x)$ with respect to each input feature x_i in a small neighborhood around the input x.

2. *Integrated Gradients*(IG) [130] for an input x is defined as $g_{IG}(x) = (x - \bar{x}) \times \int_0^1 \frac{\partial f(\bar{x} + \alpha(x - \bar{x}))}{\partial x} d\alpha$, where $\bar{x}$ is a baseline input representing the absence of features in the explained input x. The usage of IG as an explanation is to overcome the gradient saturation problem.

3. *Guided Backpropagation* (GB) [128] is the gradient explanation in which the negative gradients are set to zero when going through ReLU activations.

4. *GradCAM (GCam)* [121] provides visual explanations for predictions of CNN-based models. The method uses the gradients of any target class with respect to the feature map at the last convolutional layer to compute a coarse localization map highlighting important areas in the input image. GradCAM can be combined with Guided Backpropagation, called *Guided GradCAM* to provide pixel-level explanations.

5. *SmoothGrad* [127] averages explanations of noisy copies of an input to alleviate noise and visual diffusion [131]: $g_{smooth}(x) = 1/n \sum_{i=1}^{n} g(x + z_i)$, where z_i are drawn from a normal distribution.

6. *Layerwise Relevance Propagation* (LRP) [9] operates by back-propagating the prediction with a set of purposely designed rules. The idea is to find a Relevance score R_d^l for each neuron d at a given layer. An example computation of R_d^l is via decomposition:

$$R_j^{(l+1)} = \sum_{i \in (l)} R_{i \leftarrow j}^{(l,l+1)}, \quad R_i^{(l)} = \sum_{j \in (l+1)} R_{i \leftarrow j}^{(l,l+1)}, \quad R_{i \leftarrow j}^{(l,l+1)} = \frac{w_{ij} x_i}{\sum_k w_{kj} x_k} R_j^{(l+1)}$$

where $R_{i \leftarrow j}^{(l,l+1)}$ is the relevance score sent from neuron j of layer $l+1$ to neuron i of layer l.

7. *LIME* [111] is a model-agnostic explanation method. LIME explanation h is a linear model whose coefficients are the importance score of the features:

$$\arg\min_{h \in \mathcal{H}} \mathcal{L}(f, h, \pi_x) + \Omega(h) \tag{1-1}$$

where $\mathcal{L}$ is a loss between the explained function f and the explanation function h, $\mathcal{H}$ is the class of linear models and π_x is an exponential kernel defined on some distance function and $\Omega(h)$ is a function measuring the complexity of h.

8. *SHAP* explainer [86] can be considered as a generalized version of LIME. The main differences are the choices of the kernel π_x and the complexity measure $\Omega(h)$ so that the explanations satisfy certain desired properties of Game Theory, namely *Local accuracy*, *Missingness* and *Consistency* [82]. *KernelSHAP* is a black-box method based on SHAP, whose perturbed features are set to the average of some background data. On the other

hand, *GradientSHAP* is a white-box method based on SHAP, which relies on the gradient of the model computed on the explained input.

9. *DeepLIFT* [122] is a white-box method based on back-propagating the model. During its back-propagation, DeepLIFT computes the difference Δ_{x_i} between the activation of each neuron x_i to its 'reference activation' r_i and determines the contribution scores $C_{\Delta_{x_i}\Delta_y}$ accordingly. The DeepLIFT's attribution scores satisfy a *summation-to-delta* property:

$$\sum_i C_{\Delta_{x_i}\Delta_y} = \Delta_y$$

   where $y = f(x)$ is the model output and $\Delta_y := f(x) - f(r)$.

1.4 Dissertation Outline

All works presented in this dissertation are post-hoc methods, i.e., their analysis is conducted after the models' training. The works also mainly focus on the local behaviors of the explained models and, consequently, local explainers. Those methods are also known as *black-box* methods since they do not require internal information of the explained models. Table 1-1 outlines the scope of works presented in this dissertation.

Table 1-1. Scope of works

Chapter	Topic	Description	Model-agnostic	Post-hoc	Explanatory domain
2	c-Eval	Evaluation of explanation	N/A	Yes	N/A
3	EMaP	Perturbation method for explainer	Yes	Yes	N/A
4	NeuCEPT	Explainer	No	Yes	Model's neurons
5	PGM-Explainer	Explainer	Yes	Yes	Bayesian Networks
6	Unexplainable AI	Limitations of explainers	N/A	N/A	Bayesian Networks

Chapter 2 is about c-Eval, a metric, and the corresponding algorithm to evaluate explanations of feature-based local explainers. In particular, c-Eval evaluates the faithfulness of the explanations toward the model's prediction. It answers the question if only the features included in the explanation (explanatory features) are observed, how certain the model is on the explained prediction. The computation of the metric is based on the intuition that, if certain features are important to the prediction, it is difficult to change the prediction when those features are kept intact. The faithfulness of an explanation is therefore quantified by the minimum amount

of perturbation on features outside of the explanation that can alter the prediction. We further provide an analysis showing a connection between the importance of features contained in an explanation and its corresponding c-Eval in multi-class affine classifiers. For general non-affine classifiers, our experimental results based on c-Eval suggest the existence of nearly-affine decision surfaces in many modern classifiers. This observation encourages the adoption of the c-Eval metric in evaluating explanations of predictions made by a broad range of image classifiers. We also heuristically demonstrate the behaviors of c-Eval in adversarial-robust models. The experimental results show that the c-Eval computed in robust models is highly correlated with the non-robust counterpart, which strengthens and validates the applications of c-Eval.

To explain predictions of black-box models, many explanation methods rely on the perturbations of input data. Despite the strong influence of the perturbations on the explanations, very few works closely examined how to generate them. In particular, existing methods often ignore the data topology and unnecessarily distort it significantly. These distortions may considerably degrade explainers' performance since models are not trained to operate on the deformed topology. In *Chapter 3*, we introduce a novel perturbation scheme so that more faithful and robust explanations can be obtained. The study focuses on the impact of perturbing directions on the data topology. Specifically, the work shows that perturbing along the orthogonal directions of the input manifold better preserves the data topology, both in the worst-case analysis of the discrete Gromov-Hausdorff distance and in the average-case analysis via persistent homology. From those results, we introduce the EMaP algorithm, realizing the orthogonal perturbation scheme. The chapter also contains experiments demonstrating that EMaP not only improves the explainers' performance but also helps them overcome a recently developed attack against perturbation-based explanation methods.

Despite recent XAI studies, there still exist numerous unanswered questions on how neural networks generate their predictions. For example, given similar predictions of different inputs, are the underlying mechanisms generating those predictions the same? In *Chapter 4*, we discuss about NeuCEPT, a method to identify critical neurons that are important to the model's local predictions and learn their underlying mechanisms. We first formulate a critical neuron identification problem as the maximization of a sequence of mutual-information objectives. Then, a theoretical framework to efficiently solve for the critical neurons while keeping the precision under control is provided. After that, NeuCEPT heuristically learns the model's mechanisms in generating predictions in an unsupervised manner. We finally provide experiments and case studies showing that neurons identified by NeuCEPT not only have a strong influence on the model's predictions but also hold meaningful information about the model's mechanisms.

Graph Neural Networks (GNNs) have been emerging as powerful solutions to many real-world applications in various domains where the datasets are in the form of graphs such as social networks, citation networks, knowledge graphs, and biological networks [156, 160, 163]. In GNNs, the graph structure is incorporated into the learning of node representations. This complex structure makes explaining GNNs' predictions become much more challenging than in conventional models. In *Chapter 5*, we propose PGM-Explainer, a Probabilistic Graphical Model (PGM) model-agnostic explainer for GNNs. Given a prediction to be explained, PGM-Explainer identifies crucial graph components and generates an explanation in the form of a PGM approximating that prediction. Different from existing explainers for GNNs where the explanations are drawn from a set of linear functions of explained features, PGM-Explainer can demonstrate the dependencies of explained features in the form of conditional probabilities. We provide a theoretical analysis showing that the PGM generated by PGM-Explainer guarantees to include the Markov-blanket of the target prediction. We also demonstrate that the explanation returned by PGM-Explainer contains the same set of independence statements in the perfect map. The experiments on both synthetic and real-world datasets show that PGM-Explainer achieves better performance than existing explainers in many benchmark tasks.

Following the success of GNNs, Temporal Graph Neural Networks (TGNN) have been developed to integrate temporal information into the graph structure [162, 147, 91]. This variant has shown promising outcomes in domains where the data has strong correlations with time such as transportation and weather forecast. However, since TGNN inherits the black-box nature of neural networks, interpreting their predictions remains a daunting task. While many explanation methods for GNNs can be applied to explain predictions of TGNNs, there is little theoretical study on their behaviors. In particular, is there any information on the model that an explanation method cannot reveal? Motivated by that question, *Chapter 6* explores the limitations of some popular classes of perturbation-based explanation methods. By constructing specific instances of TGNNs, we show (i) Node-perturbation is not reliable for identifying the paths that carry out the prediction, (ii) Edge-perturbation cannot reliably determine all the nodes contributing to the prediction and (iii) perturbing both nodes and edges does not consistently help identify the graph components responsible for the temporal aggregation in TGNNs. Our experiments further demonstrate situations for failures of explanations can occur in both synthetic and real-world scenarios. Thus, they emphasize the importance of perturbation choices and the internal information of the model in determining faithful explanations for its predictions.

CHAPTER 2
C-EVAL: A UNIFIED METRIC TO EVALUATE FEATURE-BASED EXPLANATIONS VIA PERTURBATION

With the pervasiveness of machine learning in many emerging domains, especially in critical applications, it is becoming more and more important to understand how machine learning models generate predictions. For example, deep convolutional neural networks (CNNs) have been deployed to detect skin cancer at a level of competence comparable to dermatologists [37]. However, doctors and experts cannot rely on these predictions blindly. Providing additional intelligible explanations such as a highlighted skin region that contributes to the prediction will aid doctors significantly in making their diagnoses. Along this direction, many machine learning explainers supporting users in interpreting the predictions of complex neural networks have been proposed. Some notatble methods are SHAP [86], LIME [111], GCam [121], and DeepLIFT [122], among others [9, 128, 123, 127, 131, 114, 139, 112].

Even though it is important to evaluate the explanations, the evaluation task remains challenging [83, 67]. One major challenge is the lack of ground-truth explanations. In fact, most feature-based explanations have been evaluated only through a small set of human-based experiments which apparently does not imply a global guarantee of their quality [86, 122]. Another challenge is the diversity in the presentation/format of explanations. Fig. 2-1 shows an example of three explanations generated by LIME, GCam, and SHAP explainers for the prediction *Pembroke* made by the Inception-v3 image classifier [132]. All of them highlight the region containing the *Pembroke*; however, their formats vary from picture segments in LIME, heat-map in GCam to pixel importance-weights in SHAP. Furthermore, explainers might be designed for different objectives as there is a fundamental trade-off between the interpretability and the accuracy of explanations [111, 86]. In fact, utilizing the interpretability of the explanation might cost its consistency with the explained prediction. The diversity in presentations and the difference in objectives are great roadblocks to the evaluation of explanations.

The work in this chapter focuses on evaluating explanations of feature-based local explainers. Specifically, we evaluate the power of the explanations toward the model's prediction, i.e. if only the features included in the explanation are observed, how certain the model is on the explained prediction. We first introduce a novel metric, *c*-Eval, to evaluate the quality of explanations. The metric is based on the intuition that if certain features are important to the prediction, it is difficult to change the prediction when those features are kept intact. The faithfulness of an explanation is therefore quantified by the minimum amount of perturbation on features outside of the explanation that can alter the prediction.

We further provide an analysis showing a connection between the importance of features contained in an explanation and its corresponding *c*-Eval in multi-class affine classifiers. For general non-affine classifiers, the experimental results based on *c*-Eval suggest the existence of nearly-affine decision surfaces in many modern classifiers. This observation encourages the adoption of the *c*-Eval metric in evaluating explanations of predictions made by a broad range of image classifiers.

Additionally, we introduce the *c*-Eval plot, an approach based on *c*-Eval to visualize explainers' behaviors on a given input. Using LIME explainer as an example, we show how the *c*-Eval plot helps us gain more trust in LIME and select appropriate parameters for the explainers. We also heuristically demonstrate the behaviors of *c*-Eval in adversarial-robust models. Our results show that the *c*-Eval computed in robust models is highly correlated with the non-robust counterpart, which strengthens and validates the applications of *c*-Eval.

Despite the recent development of explainable AI works focusing on evaluating explanations of local feature-based explainers are quite limited. To our knowledge, there are two works that can be considered to be relevant to *c*-Eval: the AOPC score [117] and the log-odds score [122]. The AOPC score, which is introduced to evaluate heat-maps, is the average of the differences between the soft-outputs of the input image and those of some random perturbations. These random perturbations are generated sequentially based on the heat-maps on the input's features. One may think of extending AOPC to evaluate explainers, such as mask-form explanation LIME; however, it is ambiguous due to an absence of importance ordering. Furthermore, the AOPC requires a large number of random perturbations to generate stable evaluations while computing c-Eval is a deterministic process requiring only one perturbation per evaluation. On the other hand, Shrikumar *et. al* [122] use the log-odds score, measuring the difference between the input image and the modified image whose some pixels are erased, to evaluate explanations [122]. In this measurement, the erased pixels are chosen greedily based on the importance weights given by the explanation. Then, the explanation is evaluated based on how many erased pixels are needed to alter the original predicted label. However, the log-odds method is proposed without detailed analysis and it is only applicable to small gray-scale images, such as MNIST [79].

The rest of the chapter is organized as follows. Section 2.1 introduces notations and formulates c-Eval. Section 2.2 shows how to compute c-Eval. The relationship between c-Eval and the importance of input features is demonstrated in Section 2.3. The c-Eval plot, a visualization method based on c-Eval to examine explainers' behavior, is presented in Section 2.4.1. Section 2.4.2 includes the demonstration of c-Eval on adversarial-robust models. The heuristic evaluations on explanations to validate the usage of c-Eval are demonstrated in Section 2.5. Finally, Section 2.6 concludes the chapter.

2.1 C-Eval of Explanation

This section provides the notions and formulations of the proposed c-Eval metric. The explained neural network is referred to via its forwarding function f whose input and output are vectors $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{y} \in \mathbb{R}^m$. For a given vector $\boldsymbol{x}$, x_i is the element i^{th} of $\boldsymbol{x}$. The predicted label of the model's prediction $\boldsymbol{y} = f(\boldsymbol{x})$ is $l = \arg\max_{1 \le j \le m} y_j$. Given g_f, a feature-based local explainer on the classifier f, an explanation of prediction $f(\boldsymbol{x})$ is a subset of features/elements of $\boldsymbol{x}$, i.e. $e_{\boldsymbol{x}} = g_f(\boldsymbol{x}) \subseteq \boldsymbol{x}$. $e_{\boldsymbol{x}}$ is called the *explanatory features* and $\boldsymbol{x} \setminus e_{\boldsymbol{x}}$ the *non-explanatory features* of prediction $f(\boldsymbol{x})$ generated by g_f.

In feature-based explanations, the explainer may simply return $e_{\boldsymbol{x}} = \boldsymbol{x}$ as an explanation for prediction $f(\boldsymbol{x})$. This naive explanation can be interpreted as *because the input is $\boldsymbol{x}$ so the prediction is $f(\boldsymbol{x})$*. Note that this explanation is not desirable since it neither gives us any additional information on the prediction nor strengthens our trust on the model. A better answer is a smaller set of explanatory features that are important to the prediction. Thus, it is a common practice for explainers to impose cardinality constraints on $e_{\boldsymbol{x}}$ for more compact explanations [111, 112]. The evaluations of explanations in this chapter assume that they are all subjected to the same cardinality constraint $|e_{\boldsymbol{x}}| \leq k$ for a given integer k.

Let denote a perturbation scheme $h_{g_f} : \mathbb{R}^n \rightarrow \mathbb{R}^n$ with respect to (w.r.t) explainer g_f to be a function from the input's space $\mathbb{R}^n$ to itself. The resulted perturbation $h_{g_f}(\boldsymbol{x})$ is only allowed to be different from $\boldsymbol{x}$ on non-explanatory features of $f(\boldsymbol{x})$, i.e. $\boldsymbol{x} \setminus e_{\boldsymbol{x}}$:

$$h_{g_f}(\boldsymbol{x})_i = \begin{cases} x_i & \text{if } x_i \in e_{\boldsymbol{x}} \\ x_i + \delta_i & \text{if } x_i \notin e_{\boldsymbol{x}}, \end{cases} \tag{2-1}$$

where $\delta_i \in \mathbb{R}$ is the perturbation on component i^{th}. A perturbation $h_{g_f}(\boldsymbol{x})$ is considered successful under the p-norm constraint c if the predicted label of the model on $h_{g_f}(\boldsymbol{x})$ is different from the original predicted label on $\boldsymbol{x}$ and the p-norm difference between $\boldsymbol{x}$ and $h_{g_f}(\boldsymbol{x})$ is bounded by c:

$$\begin{aligned} &\arg\max_{1 \leq j \leq m} f(h_{g_f}(\boldsymbol{x})) \neq l \\ &||h_{g_f}(\boldsymbol{x}) - \boldsymbol{x}||_p \leq c. \end{aligned} \tag{2-2}$$

In this chapter, the p-norm difference between $\boldsymbol{x}$ and $h_{g_f}(\boldsymbol{x})$ is called the *perturbing distortion*. c-Eval is then defined as follows:

Definition 1. *An explainer g_f (or the corresponding explanation $e_{\boldsymbol{x}}$) of a prediction $f(\boldsymbol{x})$ is c-Eval if no perturbing scheme h_{g_f} can change the model prediction on $\boldsymbol{x}$ while keeping the perturbing distortion less than or equal to c.*

Based on Definition 1, a good feature-based explanation is supposed to be c-Eval with large c. Because, if the explanatory features $e_{\boldsymbol{x}}$ had a high power to the prediction $f(\boldsymbol{x})$, modifying values on the non-explanatory features $\boldsymbol{x} \setminus e_{\boldsymbol{x}}$ have negligible impact to the model's prediction. As the perturbation scheme h_{g_f} is only on non-explanatory features, h_{g_f} must make significant modifications to successfully alter the predicted label. Consequently, for a given explanation $e_{\boldsymbol{x}}$, the greatest value of c (in Eq. (2-2)) such that there is no h_{g_f} can successfully change the predicted label would imply the power of features in $e_{\boldsymbol{x}}$. That value is denoted by:

$$c_{f,\boldsymbol{x}}(e_{\boldsymbol{x}}) := \sup c \quad \text{s.t. } \nexists h_{g_f} \text{ satisfying (2-2).} \tag{2-3}$$

In short, for every $c \leq c_{f,\boldsymbol{x}}(e_{\boldsymbol{x}})$, there is no perturbation scheme on non-explanatory features that can alter the label of prediction while keeping the perturbing distortion less than c. We call $c_{f,\boldsymbol{x}}(e_{\boldsymbol{x}})$ the c-Eval of explanation $e_{\boldsymbol{x}}$.

To this point, the chapter has defined c-Eval and described the intuition on the connection between c-Eval of an explanation and the importance of the explanatory features. Based on that connection, c-Eval can be used as a quantitative metric to evaluate the representative power of neural networks' explanations. Before discussing the computation of c-Eval (Section 2.2) and strengthening the relationship between c-Eval and the power of explanatory features (Section 2.3), we want to emphasize some properties and remarks on the usage of the metric:

1. *Range of c-Eval:* When there is no element in the set of explanatory features, $c_{f,\boldsymbol{x}}(e_{\boldsymbol{x}}) = c_{f,\boldsymbol{x}}(\emptyset)$ is the minimum amount of perturbation onto all input's features to successfully change the original prediction. In this case, the successful perturbation h_{g_f} will return a perturbation known as the *minimally distorted adversarial examples* [18]. On the other hand, when explainer g_f returns all features of the input image, there is no perturbation h_{g_f} can alter the prediction's label and $c_{f,\boldsymbol{x}}(\boldsymbol{x}) = \infty$ by convention.

2. *Similar explanations' size:* This work limits the usage of c-Eval to explanations of the same or comparable sizes. The reason is an explainer can naively include a lot of unnecessary features in its explanations and trivially increase the corresponding c-Eval. However, this restriction does not prevent the usage of c-Eval in evaluating explanations of different explainers. In fact, we can always fix a compactness parameter k (number of input features, number of pixels, or number of image segments as explanatory features) and take the top-k important elements as an explanation. For most experiments in this paper, k is chosen to be 10% of the number of input features.

3. *Normalize c-Eval among inputs:* Given a compactness parameter k, for different inputs $\boldsymbol{x}$, the amount of minimum perturbing distortion resulting in successful perturbations can vary significantly depending on the raw features' values and formats. For instance, a RGB image can be encoded using 255 integer values or a range of float values between 0 and 1. Hence, for meaningful statistical results, some experiments use the normalized ratio between c-Eval of $e_{\boldsymbol{x}}$ and c-Eval of empty explanation, i.e. $C_{f,\boldsymbol{x}}(e_{\boldsymbol{x}}) = c_{f,\boldsymbol{x}}(e_{\boldsymbol{x}})/c_{f,\boldsymbol{x}}(\emptyset)$, to evaluate $e_{\boldsymbol{x}}$.

4. *The choice of the norm for c-Eval:* To our knowledge, there has been no research on which distance metric is optimal to measure the interpretability of explanations. There is also no consensus on the optimal distance metric of human perceptual similarity [18]. Because of the following reasons, the L_2-norm, i.e. $p = 2$, is considered in this work: (i) L_2-norm has been used to generate an explanation for neural networks' predictions [111], (ii) the computation of c-Eval is related to the generation of adversarial samples, whose initial work [133] used L_2-norm, and (iii) there exist efficient algorithms to minimize L_2-norm in adversarial generation [109, 29].

2.2 Computing C-Eval

Given an explanation, it is not straightforward to compute its c-Eval with the formula (2-3). Instead, we propose to solve for the successful perturbation scheme with the smallest distortion. Specifically, the c-Eval is computed based on the following equivalent definition:

$$c_{f,\boldsymbol{x}}(e_{\boldsymbol{x}}) = \inf c \quad \text{s.t. } \exists h_{g_f} \text{ satisfying (2-2).} \tag{2-4}$$

Based on (2-4), the c-Eval of explanation $e_{\boldsymbol{x}}$ can be obtained by solving for the minimum perturbation scheme h_{g_f} on non-explanatory features.

The computation of c-Eval can be summarized through an example shown in Fig. 2-2. Given an input image and an explanation for the prediction on that image, the minimal perturbing distortion successful perturbation on non-explanatory features of that image is computed by the *Perturbation* block. The c-Eval of the explanation is then approximated by the norm of the difference between the minimal distortion perturbation and the input image.

Fig. 2-2 provides an example of c-Eval's computation on an explanation of LIME explainer. The explanation in this case includes roughly 10% the total number of input pixels. After that, a perturbed instance $h_{g_f}(\boldsymbol{x})$ is generated using a modified version of Carlini-Wagner (CW) attack [18] where the perturbation is only on the non-explanatory features. Then, the c-Eval is the norm of the difference between the input image and the perturbed image. The reported c-Eval computed in the L_2-norm is 0.6297. For the sake of demonstration, we construct a "dummy square" of the same size as the LIME explanation, which includes the center region of the original image. We consider this mask as an explanation for the prediction and compute the c-Eval for it, which is 0.6154. The c-Eval of LIME is larger than that of the dummy square, i.e. the amount of perturbation required to change the prediction while fixing the explanatory features of LIME is greater. This result aligns with the expectation that a LIME explanation should be better than a dummy square in explaining the model's prediction.

For the computation of c-Eval, the only step that requires further specifications is the Perturbation step (Fig. 2-2) determining the minimum distortion perturbation on non-explanatory features $h_{g_f}(\boldsymbol{x})$. The CW attack is used for this step since it has been widely considered as the state-of-the-art algorithm generating minimal distortion adversarial samples of neural networks. In this attack, the solver searches for an optimal difference $\boldsymbol{\delta}$ minimizing $\mathcal{D}(\boldsymbol{x}, \boldsymbol{x}+\boldsymbol{\delta}) + \lambda l(\boldsymbol{x}+\boldsymbol{\delta})$, where $\mathcal{D}$ is a distance metric between the perturbation and the original input, l is a loss function such that $l(\boldsymbol{x}+\boldsymbol{\delta}) \leq 0$ if and only if the label of $\boldsymbol{x}+\boldsymbol{\delta}$ is different from the original label and $\lambda > 0$ is a constant weight used to adjust the priority of the algorithm. Note that $\boldsymbol{\delta}$ also needs to satisfy the box constraints, i.e. $\boldsymbol{x}+\boldsymbol{\delta}$ must be in a valid value range imposed by the dataset. Then, the optimal perturbation $\boldsymbol{\delta}$ is learned via gradient-descents.

A naive modification of the CW attack so that it only perturbs non-explanatory features is by blocking the backward update on explanatory features in the gradient descents. However, the rate of convergence of a such modification may reduce significantly if many δ_i components with high gradients are blocked. Unfortunately, the situation happens frequently as most existing explainers tend to include high-gradient components as explanatory features.

To overcome this issue, we first introduce the perturbation variables $\boldsymbol{\delta}_{e_{\boldsymbol{x}}} \in [0,1]^{n-|e_{\boldsymbol{x}}|}$ representing perturbations on non-explanatory features. We then use a mapping $s : [0,1]^{n-|e_{\boldsymbol{x}}|} \to \mathbb{R}^n$ that transforms the values in $\boldsymbol{\delta}_{e_{\boldsymbol{x}}}$ into $\boldsymbol{\delta}$. The mapping guarantees that for any explanatory feature i, $\delta_i = 0$. By using s, the optimization only acts on the non-explanatory features. To solve for $\boldsymbol{\delta}_{e_{\boldsymbol{x}}}$, Adam [69] optimizer is used with the following objective:

$$\mathcal{D}(\boldsymbol{x}, \boldsymbol{x} + s(\boldsymbol{\delta}_{e_{\boldsymbol{x}}})) + \lambda\, l(\boldsymbol{x} + s(\boldsymbol{\delta}_{e_{\boldsymbol{x}}})). \tag{2-5}$$

One drawback of CW attack is the high running-time complexity. However, from the perspective of c-Eval, we might not need exactly the minimal distortion perturbation to evaluate the explanations. Suppose we have an algorithm searching for successful perturbations on non-explanatory features of $\boldsymbol{x}$. If $e_{\boldsymbol{x}}$ is important to the prediction, it will be difficult for the algorithm to find successful perturbations by perturbing only on $\boldsymbol{x} \setminus e_{\boldsymbol{x}}$. Thus, the resulting distortion will be higher than that when $e_{\boldsymbol{x}}$ is not important. The intuition here is very similar to the definition of c-Eval in the previous section. The only difference is in the space of the perturbation schemes. Thus, we extend the definition of c-Eval to the "c-Eval with respect to a class of perturbing scheme $\mathcal{H}$" as follows:

Definition 2. *An explainer g_f (or the corresponding explanation $e_{\boldsymbol{x}}$) of a prediction $f(\boldsymbol{x})$ is c-Eval with respect to the class of perturbing schemes $\mathcal{H}$ if no perturbing scheme $h_{g_f} \in \mathcal{H}$ can change the model prediction on $\boldsymbol{x}$ while keeping the perturbing distortion less than or equal to c.*

Definition 2 helps us avoid the difficulty in finding the minimum-distortion perturbation scheme h_{g_f}. In particular, instead of examining all perturbation schemes satisfying the p-norm constraint within distance c, we can focus on the optimal h_{g_f} in a much smaller set of perturbation schemes $\mathcal{H}$. By narrowing down the choices of h_{g_f}, the computation of c-Eval can be tractable without much loss in performance. Specifically, we propose to focus on the set of perturbations generated by the Gradient-Sign-Attack (GSA) [50], and the Iterative-Gradient-Attack (IGA) [76] due to their low running time complexity. Given an image $\boldsymbol{x}$, GSA sets the perturbation $\boldsymbol{x}'$ as $\boldsymbol{x}' = \boldsymbol{x} - \varepsilon.\text{sign}(\nabla J_l(\boldsymbol{x}))$, where J_l is the l component of the loss function used to train the neural network and ε is a small constant. On the other hand, IGA initializes $\boldsymbol{x}'^{(0)} = \boldsymbol{x}$ and updates it iteratively as $\boldsymbol{x}'^{(i+1)} = \text{clip}_{\boldsymbol{x},\varepsilon}\left(\boldsymbol{x}'^{(i)} - \alpha.\text{sign}(\nabla J_l(\boldsymbol{x}'^{(i)}))\right)$, where the clip function ensures that $\boldsymbol{x}'^{(i)}$ is in the ε-neighborhood of the original image. To adopt GSA and IGA into the context of c-Eval where the perturbation is on non-explanatory features, we simply need to block the backward step of the gradient-descent algorithm on explanatory features.

Fig. 2-3 demonstrates the distortions generated by different attacks. The experiment setup is the same as in the experiment of Fig. 2-2. The L_2-norm of the distortions generated by GSA and IGA on LIME explanation are 1.3120 and 0.9804, respectively. The corresponding c-Eval for the dummy square are 1.2962 and 0.9696. We can see that the distortions in GSA and IGA are spreading out due to the nature of the attacks, which constitutes higher total distortions. Even though the distortions in GSA and IGA are larger than those computed by the CW attack, their results still imply that the LIME explanation is better than the dummy square and aligns with the intuition of their explaining power.

Fig. 2-4 is the scatter plot of c-Eval of 30 explanations in Inception-v3 computed by different perturbations methods. The results demonstrate the strong correlations of c-Eval computed by GSA as well as IGA to c-Eval obtained from CW attack. Based on these correlations, we will use GSA and IGA instead of CW attack to compute c-Eval for some experiments in this work due to their lower running-time complexity.

2.3 C-Eval and the Importance of Features

This section illustrates a relationship between c-Eval and the importance of features returned by local explainers. We first demonstrate this relationship in multi-class affine classifiers. We show that c-Eval determines the minimum distance from the explained data point (the input image) to the nearest decision hyperplane in a lower-dimension space restricted by the choice of explanatory features. A high c-Eval implies that the chosen explanatory features are more aligned with the minimum projection's direction, i.e., they are features determining the prediction on the data point. Then, the analysis of c-Eval is further heuristically extended to the more general non-affine classifiers.

2.3.1 C-Eval in Affine Classifiers

We now consider an affine classifier $f(\boldsymbol{x}) = \boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{b}$ where $\boldsymbol{W}$ and $\boldsymbol{b}$ are given model parameters. Given an explanation $e_{\boldsymbol{x}}$, c-Eval is the solution of the following program:

$$\begin{aligned} &\min ||\boldsymbol{\delta}||_2 \\ &\text{s.t } \exists j : \boldsymbol{w}_j^T(\boldsymbol{x}+\boldsymbol{\delta}) + b_j \geq \boldsymbol{w}_{j_0}^T(\boldsymbol{x}+\boldsymbol{\delta}) + b_{j_0}, \\ &\quad \forall i \in e_{\boldsymbol{x}}, \delta_i = 0, \end{aligned} \tag{2-6}$$

where $\boldsymbol{w}_j$ is the j^{th} column of $\boldsymbol{W}$, $j_0 = \arg\max_j f(\boldsymbol{x})$ is the original prediction and $\boldsymbol{\delta}$ is the vector of δ_i defined in (2-1).

When $e_{\boldsymbol{x}} = \emptyset$, there is no restriction on entries of $\boldsymbol{\delta}$. The optimization program (2-6) computes the distance between $\boldsymbol{x}$ and the complement of convex polyhedron P:

$$P = \bigcap_{j=1}^{m} \{\boldsymbol{x} : f_{j_0}(\boldsymbol{x}) \geq f_j(\boldsymbol{x})\}, \tag{2-7}$$

where $\boldsymbol{x}$ is located inside P. The optimal $c_{f,\boldsymbol{x}}(\emptyset)$ of (2-6) is a distance from $\boldsymbol{x}$ to the closest decision hyperplane $\mathcal{F}_j = \{\boldsymbol{x} : f_{j_0}(\boldsymbol{x}) = f_j(\boldsymbol{x})\}$ of P. For the sake of demonstration, Fig. 2-5 describes an example in 2-dimension space where $c_{f,\boldsymbol{x}}(\emptyset)$ is plotted in the green line.

For each explanatory feature in $e_{\boldsymbol{x}}$, the optimization space of (2-6) is reduced by one dimension. The optimization program (2-6) then solves for the shortest distance from $\boldsymbol{x}$ to the complement of polyhedron P in a lower dimension. A more important subset $e_{\boldsymbol{x}}$ implies more restrictive constraints on the optimization (2-6) and a larger distortion/distance $\|\boldsymbol{\delta}\|$. In 2-dimension space as depicted in Fig. 2-5, under an assumption that $\mathcal{F}_j$ is also the closest hyperplane of P to $\boldsymbol{x}$, $c_{f,\boldsymbol{x}}(e_{\boldsymbol{x}} = \{x_1\})$ is the distance from $\boldsymbol{x}$ to $\mathcal{F}_j$ when the feature x_1 is unchanged. Similarly, the c-Eval $c_{f,\boldsymbol{x}}(e_{\boldsymbol{x}} = \{x_2\})$ is the length of the blue line in the figure. In this case, allowing changing x_2 is easier to alter the original prediction j_0 than x_1, i.e. $c_{f,\boldsymbol{x}}(\{x_1\}) < c_{f,\boldsymbol{x}}(\{x_2\})$. It implies that x_2 is more important to the prediction than x_1.

To this point, we see that under the affine assumption on classier f, c-Eval of an explanation is the length of the projection from the data point to the decision hyperplanes in the space of non-explanatory features. Therefore, the explanation with high c-Eval contains features whose dimensions are more aligned with the shortest distance vector from the data point to the decision hyperplane. Thus, c-Eval reflects the importance of features in the explanation.

2.3.2 C-Eval in General Non-Affine Classifiers.

For more general classifiers, the set P in equation (2-7) describing the region of prediction j_0 is no longer a polyhedron. However, we now demonstrate some of the observations based on c-Eval suggesting that several state-of-the-art image classifiers might be nearly affine in a wide range of local predictions. The observation then validates the usage of c-Eval to evaluate explanations of those models.

The observation is based on a property of c-Eval in an affine classifier. Given an explanation $e_{\boldsymbol{x}}$, the previous section has shown that $c_{f,\boldsymbol{x}}(e_{\boldsymbol{x}})$ is the distance from $\boldsymbol{x}$ to $\mathcal{F}_j$ without changing features in $e_{\boldsymbol{x}}$. Similarly, $c_{f,\boldsymbol{x}}(\boldsymbol{x} \setminus e_{\boldsymbol{x}})$ is the distance from $\boldsymbol{x}$ to $\mathcal{F}_j$ without changing the complement of $e_{\boldsymbol{x}}$. As in the case of 2-dimension in Fig 2-5, $c_{f,\boldsymbol{x}}(\emptyset)$ is the height to the hypotenuse of the right triangle whose sides are $c_{f,\boldsymbol{x}}(e_{\boldsymbol{x}})$ and $c_{f,\boldsymbol{x}}(\boldsymbol{x} \setminus e_{\boldsymbol{x}})$. Thus, we have:

$$\frac{1}{c_{f,\boldsymbol{x}}(\emptyset)^2} = \frac{1}{c_{f,\boldsymbol{x}}(e_{\boldsymbol{x}})^2} + \frac{1}{c_{f,\boldsymbol{x}}(\boldsymbol{x} \setminus e_{\boldsymbol{x}})^2} \tag{2-8}$$

$$\leftrightarrow c_{f,\boldsymbol{x}}(\emptyset) = \frac{1}{\sqrt{1/c_{f,\boldsymbol{x}}(e_{\boldsymbol{x}})^2 + 1/c_{f,\boldsymbol{x}}(\boldsymbol{x} \setminus e_{\boldsymbol{x}})^2}}, \tag{2-9}$$

for any explanation $e_{\boldsymbol{x}}$. The expression on the right-hand-side of (2-9) is denoted by $\hat{c}_{f,\boldsymbol{x}}(\emptyset)$.

For non-linear classifiers f, equation (2-9) does not hold in general. However, if the decision surface $\mathcal{F}_j$ is nearly affine, $c_{f,\boldsymbol{x}}(\emptyset) \approx \hat{c}_{f,\boldsymbol{x}}(\emptyset)$ for all $e_{\boldsymbol{x}}$ as described in Fig. 2-6. Interestingly, this necessary condition holds for many data points of common image classifiers such as Inception-v3 [132], VGG19 [124] and ResNet50 [56]. Specifically, in experiment on Inception-v3 shown in Fig. 2-7, $c_{f,\boldsymbol{x}}(e_{\boldsymbol{x}})$ and $c_{f,\boldsymbol{x}}(\boldsymbol{x} \setminus e_{\boldsymbol{x}})$ are computed for 8×8 GCam explanations of predictions of the Inception-v3 using CW attack. Here, the number of explanatory features k in $e_{\boldsymbol{x}}$ are varied and the corresponding $\hat{c}_{f,\boldsymbol{x}}(\emptyset)$ are computed using equation (2-9). The value of $c_{f,\boldsymbol{x}}(\emptyset)$ is drawn using the purple straight-dot-line for reference. We can see that the two lines for $c_{f,\boldsymbol{x}}(\emptyset)$ and $\hat{c}_{f,\boldsymbol{x}}(\emptyset)$ are close to each other.

The experiments on VGG19 and ResNet50 are plotted in Fig. 2-8. Due to running-time complexity, we use GSA to compute the c-Eval on the same input image as for Inception-v3. Note that the L_2 distortion is computed based on the input space of each model. We can see that $\hat{c}_{f,\boldsymbol{x}}(\emptyset)$ and $c_{f,\boldsymbol{x}}(\emptyset)$ are close to each other in both models. It is interesting that different models share this same property, which encourages us to use c-Eval to evaluate explanations of those classifiers.

2.4 Beyond C-Eval

In the following Subsection 2.4.1, we introduce the c-Eval plot, which is a visualization of explainers' behavior on a given input based on c-Eval. Using examples on the LIME explainer, we demonstrate that c-Eval plot helps determine appropriate tuning parameters for the explainer and strengthens the usage of c-Eval in evaluating the importance of explanatory features. On the other hand, since c-Eval relies on the generation of successful perturbations, Subsection 2.4.2 discusses c-Eval's behaviors in adversarial-robust models. the experimental results show that c-Eval computed in adversarial-robust models is strongly correlated with its non-robust counterpart. The result implies that c-Eval is applicable in adversarial-robust models.

2.4.1 C-Eval Plot

In Section 2.1, we restrict the c-Eval analysis to explanations of similar sizes. That restriction is just for a fair comparison among explanations of different explainers. In practice, the explanation's size is normally determined based on the applications and the specific inputs. Given an explainer, by varying the explanation's size k, we obtain a sequence of explanations and their corresponding c-Eval. Therefore, on a given input image, each explainer will be associated with a sequence of c-Eval values. By plotting this sequence as a function of k, the behaviors of the explainer on that input can be captured. Then, an appropriate size for the explanation can be chosen accordingly. The resulting plot is called the c-Eval plot.

In the following experiment, c-Eval plot is used to study the LIME explainer with different numbers of samplings. Specifically, we show how the c-Eval plot reflects the impact of the number of samplings on LIME's performance and helps us determine appropriate values for that parameter. In LIME, the sampling size determines how many perturbations are conducted in finding the explanation. The higher the number, the better the explanation and the higher the running time complexity [111]. Since there is no concrete rule on how this parameter should be chosen, how can we verify that a LIME explanation is free from under-sampling errors? What is an appropriate number of explanatory features to explain the prediction?

The experiments are conducted on Inception-v3 with the input image as in the experiment in Fig. 2-2. First, the input image is divided into 100 feature segments. Then, the prediction of Inception-v3 on that input is explained using LIME with 100, 1000, 4000 and 10000 samples. The c-Eval sequences $\{c_{f,\boldsymbol{x}}(e_{\boldsymbol{x}}^k)\}_{k=1}^n$ is plotted in fig. 2-9. For some numbers of segments (the explanation's size $k = 5, 10, 20$ or 30) and for each setting of LIME (red for 100, blue for 1000, green for 4000 and purple for 10000 samples), the corresponding explanations are provided for illustration. The result shows distinct gaps in the c-Eval among different numbers of samplings. It is clear that the higher the number of samples, the higher the c-Eval. This observation is consistent with our expectation that the explanations at higher numbers of samples are better. Additionally, using c-Eval plot, it can be seen that there is not much improvement in the explanations' quality by increasing the number of samples from 4000 to 10000. This implies c-Eval can be used as a metric to support the automatic tuning of the explainer's parameters. It also helps us gain trust in LIME in the sense that, if we aim for top-5 important features among 100 features, LIME with 2000 samples might be reliable since there is not much gain in c-Eval by increasing that number from 1000 to 10000. Additionally, given the number of samples, there is a diminishing return in c-Eval after a certain number of explanatory features. For instance, if only 1000 samples are used, 20 explanatory features (roughly 20% of the input image) are enough to explain the prediction.

2.4.2 C-Eval on Adversarial-Trained Models

Since c-Eval is computed based on adversarial generation, there might exist concerns regarding the applications of c-Eval on adversarial-robust models. First, as adversarial-robust models are more resistant to perturbations, is it feasible to generate successful perturbations on robust models? Second, if we are able to obtain those perturbations, are the c-Eval values of the corresponding explanations reliable? This subsection addresses those concerns through experiments on the MNIST dataset using the LeNet classifier [78]. Specifically, it is shown that the c-Eval computed on non-robust and robust models have a strong correlation. This correlation implies that the behaviors of c-Eval are similar on non-robust and robust models.

The experiments in this subsection use Advertorch [29], a Python toolbox for adversarial robustness research, to train 3 LeNet classifiers on the MNIST dataset. The first model, denoted as the non-robust model, is trained normally. The second is alternatively trained between images from MNIST and the corresponding adversarial samples generated at each iteration. Here, the normalized L_2-norm distortion between each adversarial sample and its original image is bounded by $\varepsilon = 0.3$. The third model is trained in the same manner as the second but the distortion bound ε is set to 0.5. All three classifiers archive more than 95% accuracy on the test set. For the two adversarial-trained models, their accuracies on adversarial samples are all greater than 94%.

In the experiments, 4000 predictions of images from the test set are made by each LeNet. The corresponding top-10% LIME explanations are then computed for each prediction. For all explanations, the successful perturbations and the corresponding c-Eval are computed via IGA. The successful perturbations for adversarial-trained models can be computed because the models are only robust against adversarial with bounded distortion. In c-Eval, the perturbations are not limited by the amount of distortion.

Fig. 2-10 is the scatter plot of c-Eval of 300 (randomly chosen from 4000) explanations of predictions from the three classifiers. The Pearson correlations between c-Eval of the first model and those of the other two adversarial-trained models are 0.765 and 0.764, respectively. This is a fairly high correlation if we take into account the fact that the three models are trained separately. Especially when less than 75% of the explanatory features are shared between the non-robust model and any of the others. It can also be observed that the model with a higher bound in the distortion in the training has a higher average c-Eval. This aligns with our intuition on how c-Eval is computed.

2.5 Experimental Results

In this section, c-Eval is used to heuristically evaluate explanations generated by feature-based local explainers on MNIST [79] and Caltech101 [40]. Experimental results also demonstrate that the evaluations of explanations based on c-Eval align with previous evaluations conducted by the log-odd scoring function [122], which is a metric designed specifically for MNIST. To demonstrate the statistic behavior of c-Eval on large number of samples, the reported c-Eval is not precisely $c_{f,\boldsymbol{x}}(e_{\boldsymbol{x}})$ but the ratio of $c_{f,\boldsymbol{x}}(e_{\boldsymbol{x}})$ over $c_{f,\boldsymbol{x}}(\emptyset)$. This ratio is indicated by the notation $\mathrm{C}(e_x)/\mathrm{C}_0$ in the legend of each figure. The ground-truth quality rankings of explanations are obtained from previous results in assessing the explainer's performance using human-based experiments [86, 122]. The studied classifier models and explainers are selected based on those previous experiments accordingly. The system specifications and the codes for our implementations are specified in subsection 2.5.1. For reference, we also provide experiments on small-size color image dataset CIFAR10 [74], which can be found in Appendix A. In Appendix B, we also provide some explanations of images from MNIST, Caltech101, and CIFAR10 datasets along with their computed c-Eval for visualization.

2.5.1 System Specifications and Source Code

The experiments in this work are conducted in Python. The computing platform is a Linux server equipped with two Intel Xeon E5-2697 processors supporting 72 threads. The system memory comprises twelve 32 GB DDR4 sticks, each operating at 2400 MHz.

2.5.2 Simulations on MNIST Dataset

For the experiments with MNIST dataset [79], 8 different feature-based local explainers are studied: LIME [111], SHAP [86], GCam [121], DeepLIFT (DEEP) [122], Integrated Gradients [130], Layerwise Relevance Propagation (LRP) [9], Guided-Backpropagation (GB) [128] and Simonyan-Gradient (Grad) [123]. Their descriptions are provided in Section 1.3.

Since SHAP is a generalized version of LIME, we expect SHAP explanations to be more consistent with the classifier than LIME. This implies that SHAP's c-Evals are expected to be higher statistically. Previous work [86] also provided human-based experiments to support this hypothesis. DeepLIFT, Integrated Gradients, LRP, GB, and Grad are backward-propagation methods to evaluate the importance of each input neuron to the final output neurons of the examined classifier. Previous experiment results using the log-odds function in [122] suggest that GB and Grad perform worse than the other three in MNIST. GCam is an image explainer designed specifically for convolutional neural networks (CNN). Since it is not designed for classifiers of low-resolution images, it is expected that GCam's performance and its corresponding c-Eval in the MNIST dataset are limited.

The experiments on the MNIST dataset are conducted in a pixel-wise manner, i.e. the outputs of explainers are image pixels. For each input image, each explainer except LIME is set to return 10% the number of image pixels as an explanation. For LIME, since the algorithm always returns image segments as explanations, we set the returned pixels to be as close to 10% of the total number of pixels as possible. On the other hand, the implementations of LRP are simplified into Gradient$\times$Input based on the discussion in [122]. The c-Eval and the statistical results of explanations are reported in Fig. 2-11.

Different classifiers: Figs. 2-11a and 2-11b show the distributions of c-Evals on 1000 explanations in MNIST dataset on classifier 1 provided by [86] and classifier 2 provided by [122]. The notation I5 and I10 indicate the Integrated-Gradient method with 5 and 10 interpolations [130]. It can be seen that the evaluation based on c-Eval is consistent between classifiers as well as previous attempts of evaluating explainers in [86] and [122]. For the consistency in the behavior of c-Eval and log-odds function in [122], please see the discussion in subsection 2.5.4.

Different gradient-based perturbation schemes: Figs. 2-11c and 2-11d demonstrate the usage of IGA instead of GSA as shown in Figs. 2-11a and 2-11b. Comparing the distributions in Fig. 2-11c to Fig. 2-11a and Fig. 2-11d to Fig. 2-11b, the relative c-Eval of explainers are similar between perturbation schemes and consistent with previous experiments in Fig. 2-4. Thus, the computed c-Evals using IGA also reflect the explainers' performance. Finding optimal perturbation schemes resulting in a good measurement of c-Eval is not considered in this work;

however, the experiments suggest that non-optimal perturbation schemes can be used to obtain a reasonable measurement of c-Eval.

2.5.3 Simulations on Caltech101 Dataset

For experiments on large images, we study the performance of LIME, SHAP, GCam, and DeepLIFT on 700 images in Caltech101 dataset [40] with the VGG19 classifier [124]. As LIME, SHAP, and GCam explainers are designed for medium-size to large-size images, we expect they should outperform DeepLIFT. Furthermore, the results from [86] imply SHAP should perform better than LIME. On the other hand, as GCam is designed for fully-connected convolution networks (e.g. VGG19), we expect its performance here to be much better than that in previous experiments on the MNIST dataset.

Segment-wise features are used on the Caltech101 dataset. Since the outputs of some explainers are importance weights of pixels, they are all converted into scores on subsets of image segments for fair comparison. Specifically, the importance weight of each segment is the sum of the importance weights of all pixels inside that segment. Then, the top k segments with maximum weight are selected as the segment-wise explanation of the examined explainer. For the results in Fig. 2-12, the number of segments is selected such that roughly 20% of the original input image is covered by the explanation.

The computed c-Eval in the experiments on Caltech101 are reported in Fig. 2-12a and Fig. 2-12b. Here, GSA and IGA are used to compute c-Eval, respectively. The observation is that the statistical behavior of c-Eval aligns with the expectation on the performance of all four explanation methods on this dataset. In fact, by design, GCam is expected to perform better in Caltech101 than MNIST. On the other hand, DeepLIFT is not designed and stress-tested on larger models trained on datasets that are significantly bigger than MNIST. More will be discussed in Subsection 2.5.5. For the improvement of GCam and the degradation of DeepLIFT from the MNIST dataset and CIFAR10 to the Caltech101 dataset, For better intuition on the improvement of GCam and the degradation of DeepLIFT from the MNIST dataset and CIFAR10 to the Caltech101 dataset, we suggest readers check the examples shown in Appendix B.

2.5.4 Similarity of C-Eval and Log-Odds Functions in MNIST

To evaluate importance scores obtained by different methods on the MNIST dataset, the authors of DeepLIFT designs the log-odds function as follows. Given an image that originally belongs to a class, they identify which pixels to erase to convert the original image to another target class and evaluate the change in the log-odds score between the two classes. The work conducted experiments of converting 8 to 3, 8 to 6, 9 to 1 and 4 to 1. In Fig. 2-13, we adopt *c*-Eval into the MNIST dataset to compare *c*-Eval of explainers with the corresponding log-odds scores. The figure displays the *c*-Eval of studied explainers on images with predictions $4, 8$, and 9 respectively. The experiments are conducted using both GSA and IGA perturbation schemes. Besides the DeepLIFT in experiments for labels 4 and 8, all relative ranking of explainers in *c*-Eval is consistent with the ranking resulting from log-odds computations shown in [122]. This result implies that our general frameworks of evaluating explainers based on *c*-Eval are applicable to this specific study on the MNIST dataset.

2.5.5 Overall Evaluations of Explanations Using C-Eval

Many interesting results and deductions can be drawn from experiments on MNIST and Caltech101. The results are also consistent with our experiments on CIFAR10 reported in Appendix A. The following subsection discusses several key observations from the experiments.

The first observation is about the correlation of *c*-Eval and the portion of the predicted object captured by different explanations. In CIFAR10 and especially Caltech101 (see Appendix A), it is clear that most explanations containing the essential features of the predicted label have high *c*-Eval.

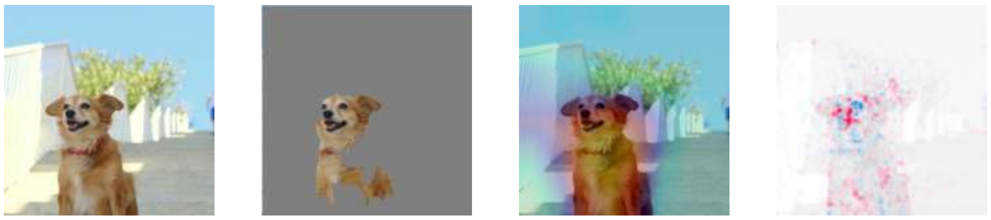
The second remark is on the relative performance of GCam in three datasets. Since GCam is designed for convolutional neural networks such as the VGG19, high relevant explanations are expected for GCam in experiments on Caltech101. However, as GCam exploits the last layer of the neural networks to generate the explanations [121], the expectation for GCam on MNIST and CIFAR10 dataset is lower. The reason is that the models used in those two later datasets are significantly different from VGG19. In fact, the adaptation of VGG19 on CIFAR10 [84] contains only 4 neurons in the last convolutional layer, which results in only 4 regions of the images that GCam can assign importance scores. The distributions of c-Eval of GCAM shown in Fig. 2-12b further reflect that expectation.

DeepLIFT is a back-propagation method and it is not only sensitive to the classifier structure but also the selection of reference image [122]. The experimental setups of DeepLIFT in the MNIST dataset shown in Fig. 2-11 are taken directly from the source code of the explainer's paper. Our adoptions of DeepLIFT to CIFAR10 and Caltech101 are conducted without calibration on the reference image as the calibration procedure for color images is not provided. This might be the reason for the degradation of the explainer's quality in these two datasets. It is clear that c-Eval captures this behavior.

It is noteworthy to examine the exceptionally high c-Eval of SHAP shown in all three datasets. This result encourages us to take a closer look at explanations produced by SHAP. It is shown via examples that SHAP captures some important features that are overlooked by others. Let's consider the explanation of number 4 as an example. SHAP is the only explainer detecting that the black area on top of number 4 is important (see examples in Appendix B). In fact, this area is essential to the prediction since, if these pixels are white instead of black, the original prediction should be 9 instead of 4. Without the c-Eval computations, it is non-trivial to identify this beneficial behavior in the SHAP explainer.

2.6 Conclusion

In this chapter, we introduce c-Eval to evaluate explanations of various feature-based explainers. Extensive experiments show that c-Eval of explanation reflects the importance of features included in the explanation. This study leads to several interesting research questions for future work. For example, the distributions of c-Eval in Fig. 2-11 advocate that there is a fundamental difference between the quality of black-box explainers (SHAP, LIME, and GCam) and back-propagation explainers (DEEP, Integrated Gradients, LRP, GB, and Grad), which is ambiguous prior to this work.



(a) Original (b) LIME (c) GCam (d) SHAP

Figure 2-1. Explanations generated by three feature-based local explainers for the prediction *Pembroke* of Inception-v3 model.

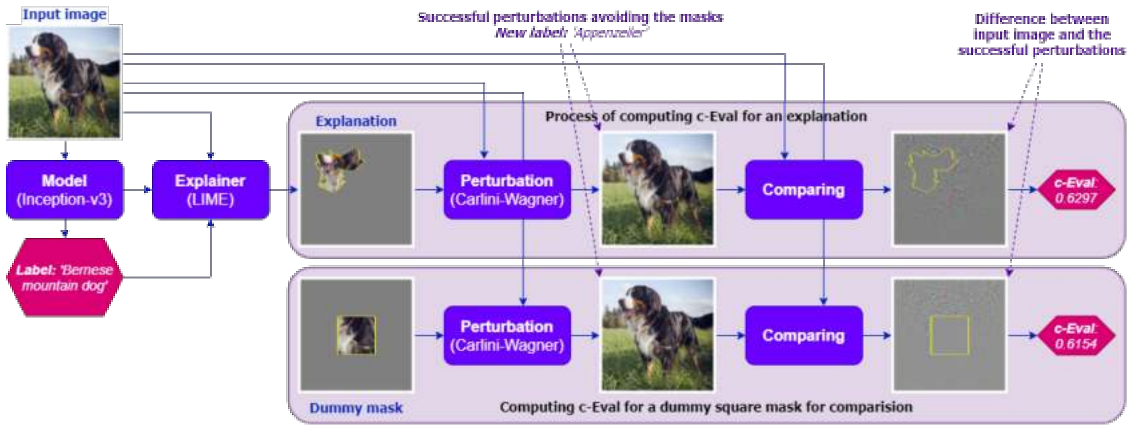
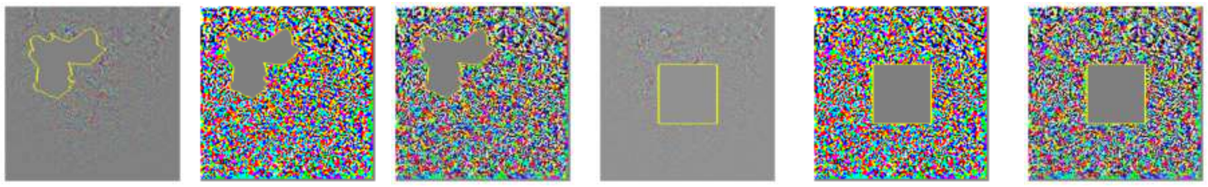

Input image
Successful perturbations avoiding the masks
New label: 'Appenzeller'
Difference between input image and the successful perturbations
Model (Inception-v3)
Explainer (LIME)
Label: 'Bernese mountain dog'
Explanation
Process of computing c-Eval for an explanation
Perturbation (Carlini-Wagner)
Comparing
c-Eval: 0.6297
Perturbation (Carlini-Wagner)
Comparing
c-Eval: 0.6154
Dummy mask
Computing c-Eval for a dummy square mask for comparision

Figure 2-2. Example comparing the importance of features included in an explanation and features in a dummy mask using *c*-Eval.



(a) LIME-CW (b) LIME-GSA (c) LIME-IGA (d) Dummy-CW (e) Dummy-GSA (f) Dummy-IGA

Figure 2-3. Distortions between perturbations and the original images. The notations 'LIME' and 'Dummy' stand for LIME explanation and dummy explanation in the experiment shown in Fig. 2-2.

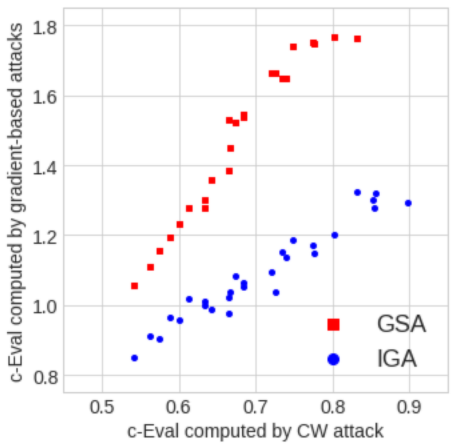


Figure 2-4. Scatter plot of c-Eval computed by gradient-based attacks vs c-Eval computed by CW attack.

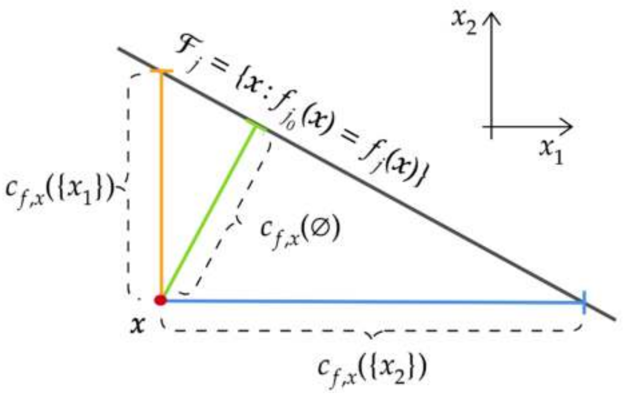


Figure 2-5. c-Eval in 2D affine classifier. Explanation $\{x_2\}$ is better than $\{x_1\}$ since the distance from $\boldsymbol{x}$ to hyperplane $\mathcal{F}_j$ without changing x_2 is larger than that distance without changing x_1.

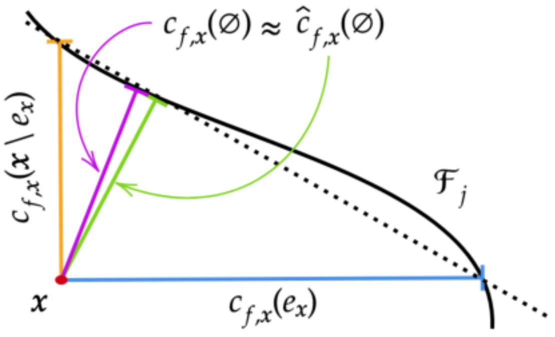

$c_{f,x}(\varnothing) \approx \hat{c}_{f,x}(\varnothing)$
$c_{f,x}(x \setminus e_x)$
$\mathcal{F}_j$
x
$c_{f,x}(e_x)$

Figure 2-6. c-Eval in non-linear classifier. When f is nearly affine, $c_{f,\boldsymbol{x}}(\emptyset) \approx \hat{c}_{f,\boldsymbol{x}}(\emptyset)$.

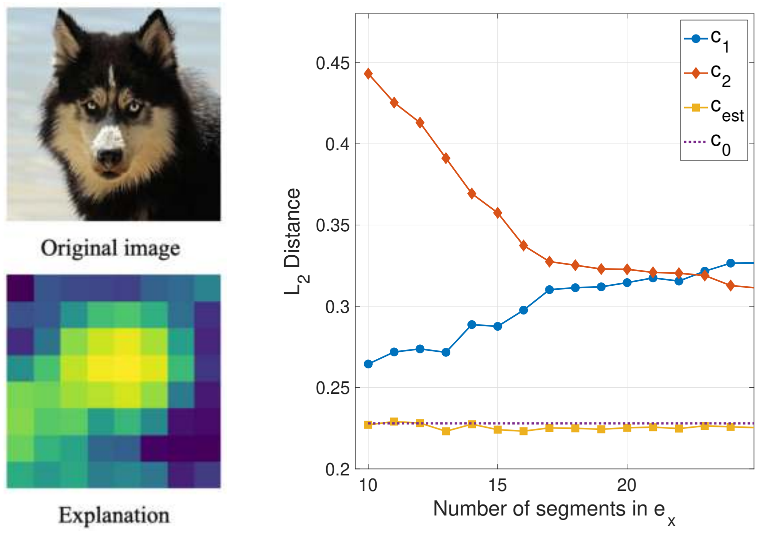
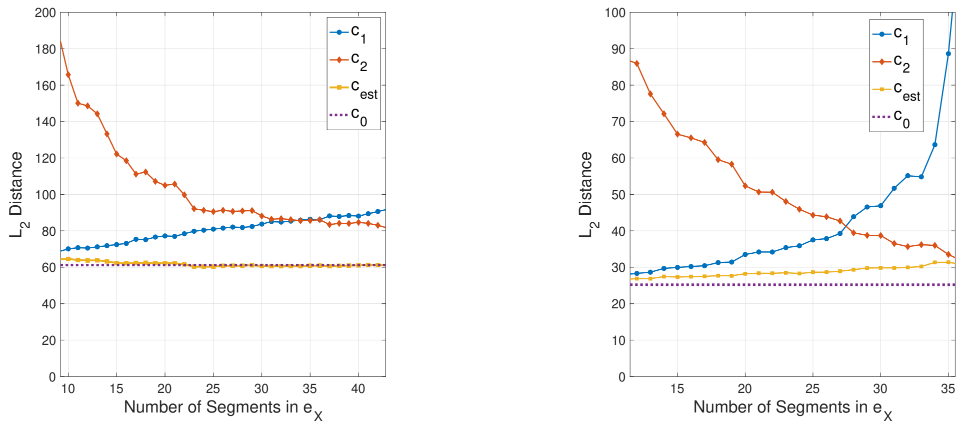


Figure 2-7. Example of nearly affine instance on Inception-v3. Here, c_1, c_2, c_{est} and c_0 are $c_{f,\boldsymbol{x}}(e_{\boldsymbol{x}}), c_{f,\boldsymbol{x}}(\boldsymbol{x} \setminus e_{\boldsymbol{x}}), \hat{c}_{f,\boldsymbol{x}}(\emptyset)$ and $c_{f,\boldsymbol{x}}(\emptyset)$ respectively. Since $c_{f,\boldsymbol{x}}(\emptyset) \approx \hat{c}_{f,\boldsymbol{x}}(\emptyset)$ for all number of segments from the explanation, we might infer that the decision surface is nearly affine in this example.



(a) VGG19 (b) ResNet50

Figure 2-8. Demonstration of the condition $c_{f,\boldsymbol{x}}(\emptyset) \approx \hat{c}_{f,\boldsymbol{x}}(\emptyset)$ in VGG19 and ResNet50, which suggests the existence of nearly-affine decision surfaces in these models.

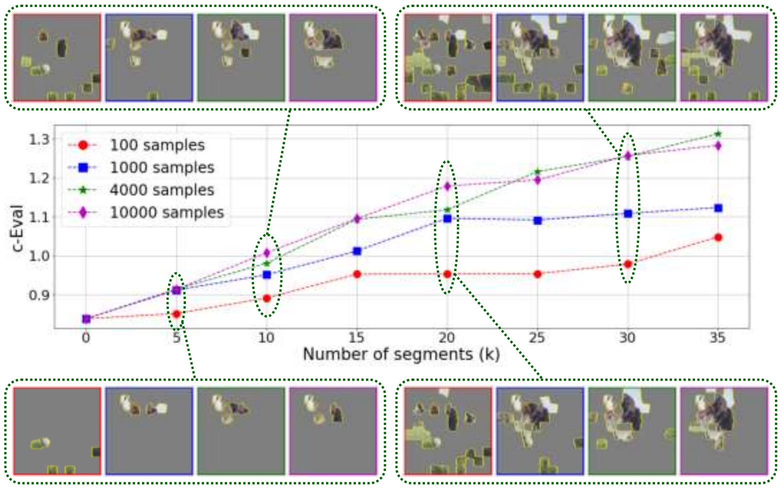


Figure 2-9. *c*-Eval plot of LIME with different sample rates. Higher sample rates result in better explanations and *c*-Eval plot reflects that expectation.

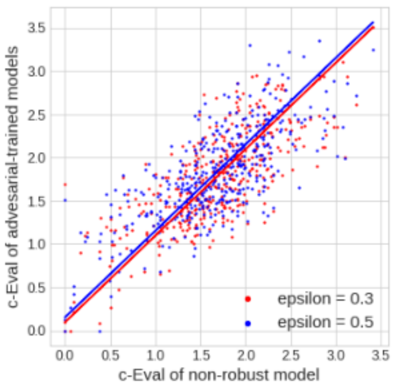
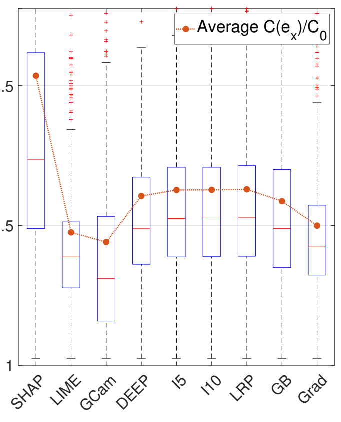
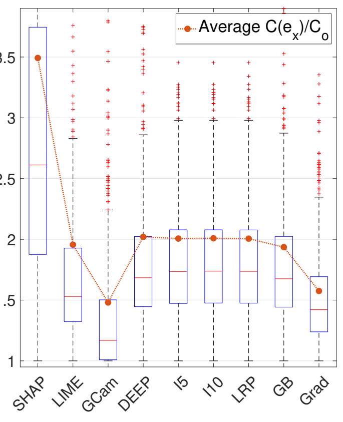


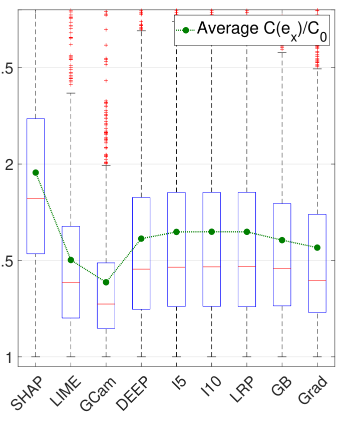
Figure 2-10. Correlation between *c*-Eval on non-robust and adversarial-trained models.



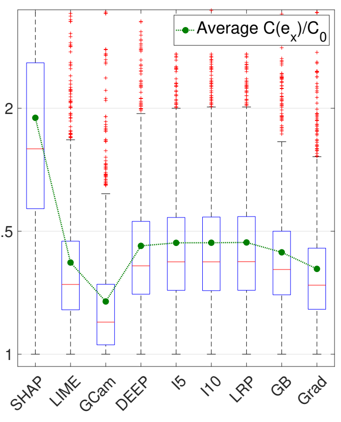
(a) GSA on classifier 1.



(b) GSA on classifier 2.

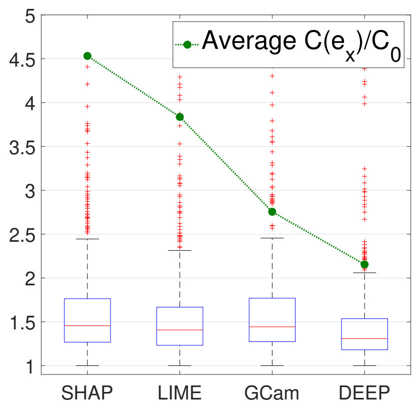


(c) IGA on classifier 1.

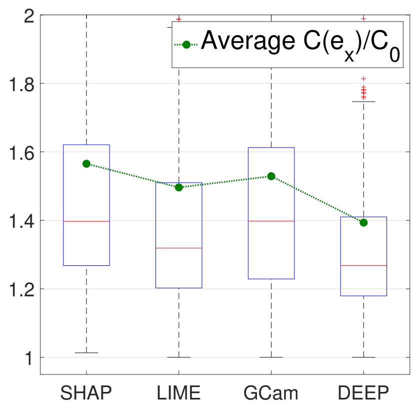


(d) IGA on classifier 2.

Figure 2-11. The distributions and the averages of *c*-Eval of 8 explainers on classifier 1 provided by [86] and on classifier 2 provided by [122] on 1000 images in MNIST dataset.

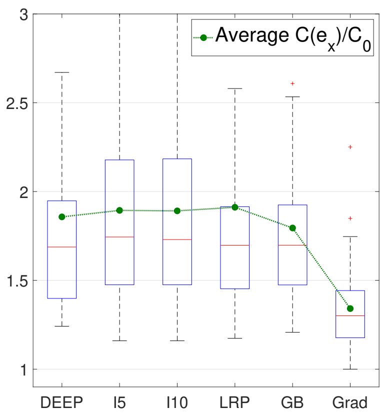


(a) GSA.

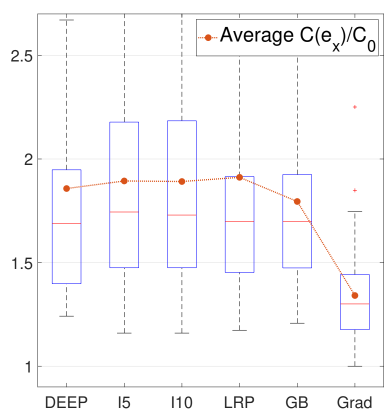


(b) IGA.

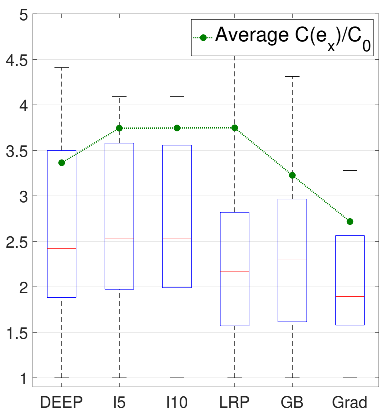
Figure 2-12. Distributions of *c*-Eval computed by GSA and IGA for four explainers in Caltech101 dataset.



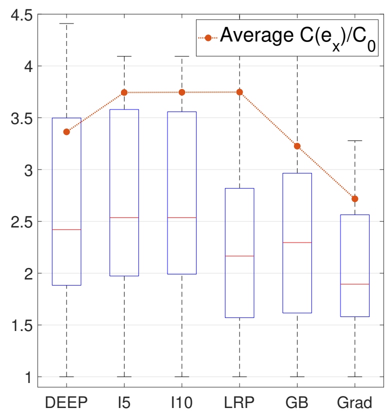
(a) *c*-Eval of label 4 with GSA



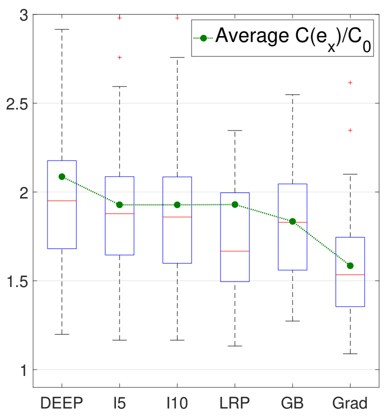
(b) *c*-Eval of label 4 with IGA



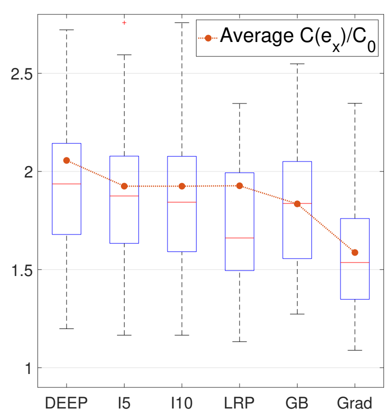
(c) *c*-Eval of label 8 with GSA



(d) *c*-Eval of label 8 with IGA



(e) *c*-Eval of label 9 with GSA



(f) *c*-Eval of label 9 with IGA

Figure 2-13. We compute the *c*-Eval for 6 explainers on 1000 images of MNIST for labels 4, 8 and 9 to show the similarity between *c*-Eval and log-odds function in [122].

CHAPTER 3
EMAP: EXPLAINABLE AI WITH MANIFOLD-BASED PERTURBATIONS

To explain black-box models, many explainers rely on some perturbations of the explained input [129, 111, 86]. For that reason, these methods are also known as *perturbation-based explainers*. Despite the strong impact of the perturbations on the pexplanations [111, 86], very few works closely examined this step. Current perturbation schemes often ignore the data topology and distort it significantly as a result. These distortions may considerably degrade explainers' performance since models are not trained to operate on the deformed topology. Additionally, the difference between the perturbations and the original data creates opportunities for malicious intents. For example, the work [125] demonstrates that a discriminator trained to recognize the explainer's perturbations can be exploited to fool the explainer.

Motivated by that lack of study, our work aims to re-design the perturbation step in the explanation process. We begin with the observation that there exists important topological information of the data having strong influences on the model trained on that data. For example, training a neural network to distinguish points from two separated point clouds is generally easier than on two overlapping/connected point clouds. Since perturbing the data can significantly change the topological information, perturbation methods can significantly change the model's behavior and influence the subsequence explaining tasks. As such, we aim to generate perturbations so that the topological structure of the original data is better preserved. Our key result is that, assuming the input data is embedded in an affine subspace whose dimension is significantly smaller than that of the data dimension, eliminating the perturbations' components along that affine subspace would better preserve the topological integrity of the original manifold. An illustration of that result is provided in Fig. 3-1, which shows perturbing along the orthogonal directions (i.e. no subspace's directions) results in smaller distortion in the topological structure of the original data. This phenomenon is also reflected in the smaller Bottleneck distances in dimensions 0 and 1, denoted by H_0 and H_1.

Based on that result, this chapter further proposes a novel manifold-based perturbation method aiming to preserve the topological structure of the original data, called EMaP. The high-level operations of EMaP are shown in Fig. 3-2. Given some sampled data, EMaP first learns a function mapping the samples to their low-dimensional representations in the data subspace. Then, that function is used to approximate a local affine subspace, shortened to local-subspace, containing the data in the neighborhood of the input to be explained. Finally, the EMaP perturbations are generated by adding the noise vectors that are orthogonal to that local-subspace to the data.

Our contributions in this work are: **(a)** This chapter theoretically shows that the worst-case discrete Gromov-Hausdorff distance between the data and the perturbations along the manifold's directions is larger than that along the orthogonal directions. **(b)** The worst-case analysis suggests that eliminating perturbation's components along the manifold's directions can generally better maintain the topological integrity of the original manifold, i.e. the average case. This work then provides synthetic and real-world experiments based on persistent homology and Bottleneck distance to support that hypothesis. **(c)** This work proposes EMaP, an algorithm generating perturbations along the manifold's orthogonal directions for explainers. EMaP first approximates the input's manifold locally at some given data points, called pivots, and the explained data point. The perturbations are then generated along the orthogonal directions of these local-subspaces. EMaP also computes the low-dimensional distances from the perturbations to the explained data point so that the explainers can better explain the model. **(d)** Finally, experiments on four text datasets, two tabular datasets, and two image datasets, are provided. The results show that EMaP can improve the explainer's performance and protect explainers from adversarial discriminators.

The remainder of the chapter is structured as follows. Sections 3.1 and 3.2 briefly discuss related work and preliminaries. Section 3.3 presents the analysis of the discrete Gromov-Hausdorff distances of different perturbation directions, which suggests orthogonal directions are preferable. That result is then strenthen with a persistent homology analysis in Section 3.4. Sections 3.5 and 3.6 describe the proposed EMaP algorithm and its experimental results. Section 3.7 concludes the chapter.

3.1 Related Work

This work intersects several emerging research fields, including explainers and their attack/defense techniques. Our approach also uses recent results in topological data analysis. We provide an overview of the related work below.

Perturbation-based explanation methods: The usage of perturbations is popular among explanation methods for black-box models since it does not require any knowledge of the explained model. Notable ones are LIME [111], SHAP [86], and some others [129, 159, 131, 21, 120, 85, 155, 140]. While those methods share the same goal to explain the model's predictions, they are not only different in the objectives but also in the perturbation schemes: some zero out features [159, 120] or replace features with neutral values [111, 131], others marginalize over some distributions on the dataset [111, 86, 85]. There also exist methods relying on separate models to generate perturbations [129, 21]. More detailed descriptions of those methods can be found in [25].

Adversarial attack on explainers: This chapter studies the attack framework [125], in which the adversary intentionally hides a biased model from the explainer by training a discriminator to recognize its query. The framework will be discussed in detail in Section. 3.2. There are other emerging attacks on explainers focusing on modifying the model's weights and tampering with the input data [46, 30, 58, 28].

Defense techniques for perturbation-based explainers: Since most attacks on perturbation-based explainers were only developed recently, defense techniques against them are quite limited. Existing defenses generate perturbations either from carefully sampling the training data [20] or from learning some generative models [116, 138]. The advantage of EMaP is that it does not require any generative model, which not only reduces the attack surface but also allows theoretical study of the perturbations.

Topological Data Analysis: Topological data analysis (TDA) is an emerging field in mathematics, applying the techniques of topology (which was traditionally very theoretical) to real-world problems. Notable applications are data science, robotics, and neuroscience. TDA uses deep and powerful mathematical tools in algebraic topology to explore topological structures in data and to provide insights that normal metric-based methods fail to discern. The most common tool in the TDA arsenal is persistent homology, developed in the early 2000s by Gunnar Carlsson and his collaborators. Readers can refer to [48, 35] for an overview of both persistent homology and TDA as a whole.

3.2 Preliminaries

This section discusses the notations and some preliminaries for this chapter. We use the standard setting of the learning tasks where the set of input X is sampled from a distribution on $\mathbb{R}^N$. X is also assumed to be in a manifold embedded in an affine subspace $\mathbb{R}^V$, where V is much smaller than N. We also consider a black-box classifier f mapping each input $x \in X$ to a prediction y, a local explainer g, a (adversarial) discriminator $\mathcal{D}$, and a masking-model f'.

In the following, we describe how explainers and the explanation process are formulated. After that, we provide two examples demonstrating how perturbations along different directions can have significant impacts on the explaining process. We end this section with an attacking framework on perturbation-based explainers and describe how better perturbations can help defend against that class of attacks.

3.2.1 Explainers

An explanation of the prediction $y = f(x)$ can be obtained by running an explainer g given the input x and the model f. The resulting explanation is denoted by $g(x)$. In additive feature attribution methods, the range of $g(x)$ is a set of features' importance scores. The focus of analysis is the class of perturbation-based explanations whose importance scores are computed based on the model's predictions of some perturbations of the input. Typically, the perturbations are generated by perturbing x or/and some samples in X. The perturbations are denoted by X_r, where $r \geq 0$ specifies the perturbation's norm. A more rigorous definition for this notation is discussed in Section 3.3.

Given the perturbations, the explainer typically finds the explanation g such that its behavior on the perturbations is similar to that of the explained function f. That is the reason why they are also known as *surrogate methods*. For example, LIME explanation is a linear model whose coefficients are the importance score of the features:

$$\arg\min_{g\in\mathcal{G}} \mathcal{L}(f, g, \pi_x) + \Omega(g) \tag{3-1}$$

where $\mathcal{L}$ is a loss between the explained function f and the explanation function g, $\mathcal{G}$ is the class of linear models and π_x is an exponential kernel defined on some distance function and $\Omega(g)$ is a function measuring the complexity of g. For SHAP explainer, the main differences are the choices of the kernel π_x and the complexity measure $\Omega(g)$ so that the explanations satisfy certain desired properties of Game Theory. For details of the two methods, we refer readers to the original papers [111, 86].

3.2.2 Topological-Awareness Perturbations for Explaining Tasks

We now provide two examples demonstrating how leveraging topological information of the data can be beneficial for the explanation tasks. The examples aim to demonstrate two key observations. The first is that the usage of only projecting perturbations, i.e., perturbation in the manifold, might lead to high explaining errors. The second example heuristically shows orthogonal perturbations resulting in a more desirable model response for the explaining task.

In the first example, we consider 2-dimensional data lying in a 1-dimensional manifold (Fig. 3-4). The task of the model is to differentiate the blue data from the green data. The decision boundary of the model is a line intersecting the manifold (the red dashed line). The goal of perturbation-based explainers is to determine that decision boundary. Fig. 3-4a demonstrates the scenario when only projecting perturbations are used. Since the perturbations and the data lie in the 1-dimensional manifold, they cannot help the explainer differentiate the decision boundary from any other lines going through the intersection. Thus, the resulting explanation (purple dot line) might have high errors. On the other hand, orthogonal perturbations can differentiate lines in the 2-dimensional space and help the explainer learn solutions with lower errors (Fig. 3-4b).

Our second example is the experiment shown in Fig. 3-3. The experiment consists of a C-shaped dataset and a regression model f mapping the dataset uniformly to the range $(0, 1)$, i.e.,

$f(x) = \theta_x/(2\pi - 0.5)$ for $\theta_x \in [0, 2\pi - 0.5]$, where θ_x is the angular coordinate of the point x on the shape. That angle θ_x, the mapping f, and the dataset are illustrated in Fig. 3-3a. The first observation is that the optimal linear local explanation (1-1) of this regression model is the tangent line at the point of prediction. To obtain that explanation, one necessary condition of the perturbations is that the following quantity $f(x+\delta) - f(x) = \frac{\theta_{x+\delta} - \theta_x}{2\pi - 0.5}$ should depend only on δ but not x. In other words, the change $f(x+\delta) - f(x)$ should be almost constant when x moves along the shape. Particularly, Fig. 3-3b demonstrates the low variance of that quantity (black line) when the perturbation $x+\delta$ is sampled from the original C-shaped data, i.e., in distribution[1].

For projection and orthogonal perturbations, we can observe that $f(x+\delta) - f(x)$ matches that of in-distribution perturbation, except for data at the two ends of the C-shape. The reason is that the two ends have important topological information on the data and that information also has a strong impact on the model: if the data is slightly perturbed at those ends, the data might become connected (a circle appears) and the two extreme values of the regression model will also be connected. Thus, the evaluations of the model on the perturbations connecting the two ends, i.e. on those that destroy the associate topological feature, will result in unstable responses. Consequently, explaining the model based on those perturbations is not desirable. In fact, the distortion $f(x+\delta) - f(x)$ caused by projection perturbations has significantly higher fluctuation than that of orthogonal. The reason is that projection perturbations near the two ends of the C-shaped have a higher chance of connecting the ends and destroying the shape.

3.2.3 Attack Framework

The discriminator-based attack framework introduced by [125] is illustrated in Fig. 3-5. The framework focuses on an adversary with an incentive to deploy a biased-model f. This adversary bypasses the detection of the explainers by forwarding the explainer's perturbations to a masking model f'. The decision of whether to forward an input to the masking model is made by a discriminator. Thus, the success of the attack is determined by the capability of the discriminator

1 To sample the perturbation $x+\delta$, we first fix a perturbation magnitude Δ, then randomly select the point x' in the specified point cloud (in-data, projection or orthogonal perturbations) such that the L_2 distance between x' and x equal to Δ, i.e., $\|\delta\|_{L_2} = \Delta$. The value of Δ for this experiment is 0.25.

to distinguish the perturbations from the actual data. Thus, if the explainer can craft an X_r similar to X, it not only improves the explainer's performance but also prevents adversarial exploitations.

3.3 Analysis of Discrete Gromov-Hausdorff Distances of Perturbations

The upcoming analysis considers the following perturbation problem: Given a manifold embedded in $\mathbb{R}^N$, how do we perturb it so that we preserve as much topological information as possible? More concretely, given a finite set of points sampled from such a manifold, is there a consistent method to perturb the original dataset while preserving some notion of topology?

To begin talking about differences between (metric) spaces, we need to introduce a notion of distance between them. One such commonly used distance is the **Gromov-Hausdorff distance**, which is derived from the **Hausdorff distance**. Intuitively, a small Gromov-Hausdorff distance means that the two spaces are very similar to metric spaces. Thus, we can focus our study on the Gromov-Hausdorff distances between the data and different perturbation schemes. However, as it is infeasible to compute the distance in practice, it is more beneficial to study an approximation of it, which is the ***discrete*** **Gromov-Hausdorff distance**. Specifically, we show that, when the perturbation is significantly small, the worst-case discrete Gromov-Hausdorff distance resulting from the orthogonal perturbation is smaller than that of projection perturbation, i.e., perturbation along the manifold (Theorem 3-1). The proof of that claim relies on Lemma 3-1, which states that, with a small perturbation, the discrete Gromov-Hausdorff distance between the original point cloud and the perturbation point cloud equals the largest change in the distances of any pair of points in the original point cloud. With the Lemma, the problem of comparing point clouds is further reduced to the problem of comparing the change in distances.

The structure of this section is as follows: first, we recall the definition of the Hausdorff distance and the Gromov-Hausdorff distance. We then introduce the discrete Gromov-Hausdorff distance, which will be the main focus for computational purposes. Finally, using such discrete distance, we prove Lemma 3-1 and Theorem 3-1.

Let (M,d) be a metric space. For a subset $S \subseteq M$ and a point $y \in M$, the distance between S and y is given by $d(S,y) := \inf_{x \in S} d(x,y)$.

Definition 3 (Hausdorff distance)**.** *[51] Let S and S' be two non-empty subsets of a metric space (M,d). The Hausdorff distance between S and S', denoted by $d_H(S,S')$ is:*

$$d_H(S,S') := \max\left(\sup_{x\in S} d(S',x), \sup_{y\in S'} d(S,y)\right)$$

Definition 4 (Gromov-Hausdorff distance)**.** *[51] Let X,Y be two compact metric spaces. The Gromov-Hausdorff distance between X and Y is given by:*

$$d_{GH}(X,Y) = \inf_{f,g} d_H(f(X), f(Y))$$

where the infimum is taken over all metric spaces M and all isometric embeddings f and g.

Even though the Gromov-Hausdorff distance is mathematically desirable, it is practically non-computable since the above infimum is taken over all possible metric spaces. In particular, this includes the computation of the Gromov-Hausdorff distance between any two point clouds. In 2004, [98] addressed this problem by using a discrete approximation of Gromov-Hausdorff, which looks at the distortion of pairwise distances over all possible matchings between the two point clouds. Formally, given two finite sets of points $X = \{x_1, ..., x_n\}$ and $Y = \{y_1, ..., y_n\}$ in a metric space (M,d), the discrete Gromov-Hausdorff distance between X and Y is given by

$$d_J(X,Y) = \min_{\pi\in S_n} \max_{i,j} \frac{1}{2}\left|d(x_i,x_j) - d(y_{\pi(i)}, y_{\pi(j)})\right| \tag{3-2}$$

where S_n is the set of all n-permutations.

Let $X = \{x_1, ..., x_k\} \in \mathbb{R}^V \subseteq \mathbb{R}^N$ be a point cloud contained in some affine subspace $\mathbb{R}^V$ of $\mathbb{R}^N$. We say X is generic if the pairwise distances between the points in X are not all equal, i.e. there exist some points $x_{i_1}, x_{i_2}, x_{i_3}, x_{i_4} \in X$ such that $d(x_{i_1}, x_{i_2}) \neq d(x_{i_3}, x_{i_4})$.

Let X_r be a finite set of points in $\mathbb{R}^N$ s.t. for every $x_i \in X$, there exists a unique $\tilde{x}_i \in X_r$ such that $d(x_i, \tilde{x}_i) = r$. X_r realizes a perturbation of X with the radius of perturbation being equal to r. We also denote $X_r^{\perp}$ (resp. X_r^{Proj}) as a finite set of points in $\mathbb{R}^N$ such that for every $x_i \in X$, there exists a unique $\tilde{x}_i \in X_r^{\perp}$ such that $d(x_i, \tilde{x}_i) = r$ and $\overline{x_i\tilde{x}_i} \perp \mathbb{R}^V$ (resp. $\overline{x_i\tilde{x}_i} \subset \mathbb{R}^V$), where $\overline{x_i\tilde{x}_i}$ denotes the line connecting the points x_i and $\tilde{x}_i$. We are now ready to state the following key theorem:

Theorem 3-1. *Given a generic point-cloud* $X = \{x_1, ..., x_k\} \in \mathbb{R}^V \subset \mathbb{R}^N$*, there exists an* $r_0 > 0$ *such that for any* $r < r_0$ *and for any instances of* $X_r^{\perp}$*, there exists an* X_r^{Proj} *such that:*

$$d_J(X, X_r^{\perp}) \leq d_J(X, X_r^{\text{Proj}})$$

The proof of Theorem 3-1 is based on the following lemma:

Lemma 3-1. *There exists an* $\varepsilon > 0$ *such that for any* $r < \varepsilon$*, we have*

$$d_J(X, X_r) = \frac{1}{2} \max_{i,j} |d(x_i, x_j) - d(\tilde{x}_i, \tilde{x}_j)|$$

for any X_r*.*

To prove Lemma 3-1, we show that, for a small enough ε, the optimal permutation in Eq. (3-2) is the identity $\pi(i) = i$. Thus, the minimization in the computation of d_J can be eliminated. The detail is shown in the following.

Proof. of Lemma 3-1. Given a permutation $\pi \in S_n$ and two point clouds X, Y of the same cardinality, denote:

$$D_\pi(X, X_r) = \frac{1}{2} \max_{i,j} |d(x_i, x_j) - d(\tilde{x}_{\pi(i)}, \tilde{x}_{\pi(j)})|$$

Let N_0 be the set of permutations $\pi \in S_n$ such that $D_\pi(X, X) = 0$. Let $N_1 = S_n \backslash N_0$. Since X is generic, N_0 does not include all of S_n and $N_1 \neq \emptyset$. Let $\delta = \min_{\pi \in N_1} D_\pi(X, X) > 0$. We now show that, by choosing $\varepsilon = \frac{\delta}{4}$, we have the lemma.

Given an X_r, for any $\pi \in S_n$, we now consider two cases:

1. If $\pi \in N_0$, then $d(x_i, x_j) = d(x_{\pi(i)}, x_{\pi(j)}) \; \forall \; (i, j)$. We have:

$$\begin{aligned} D_\pi(X, X_r) &= \frac{1}{2} \max_{i,j} |d(x_i, x_j) - d(\tilde{x}_{\pi(i)}, \tilde{x}_{\pi(j)})| \\ &= \frac{1}{2} \max_{i,j} |d(x_{\pi(i)}, x_{\pi(j)}) - d(\tilde{x}_{\pi(i)}, \tilde{x}_{\pi(j)})| = \frac{1}{2} \max_{i,j} |d(x_i, x_j) - d(\tilde{x}_i, \tilde{x}_j)| \end{aligned} \quad (3\text{-}3)$$

Thus, the identity permutation belongs to N_0. Since $d(x_i,\tilde{x}_i) = d(x_j,\tilde{x}_j) = r$, we have:

$$d(\tilde{x}_i,\tilde{x}_j) \leq d(x_i,x_j) + d(x_i,\tilde{x}_i) + d(x_j,\tilde{x}_j) = d(x_i,x_j) + 2r$$
$$d(\tilde{x}_i,\tilde{x}_j) \geq d(x_i,x_j) - d(x_i,\tilde{x}_i) - d(x_j,\tilde{x}_j) = d(x_i,x_j) - 2r$$

Therefore, for all i, j:

$$|d(\tilde{x}_i,\tilde{x}_j) - d(x_i,x_j)| \leq 2r \tag{3-4}$$

This implies $D_\pi(X,X_r) \leq r$ for $\pi \in N_0$.

2. If $\pi \in N_1$, without loss of generality, we assume the pair (x_1,x_2) maximizes $|d(x_i,x_j) - d(x_{\pi(i)},x_{\pi(j)})|$. For convenience, we denote $\pi(1) = 3$ and $\pi(2) = 4$. From the fact that $\pi \in N_1$, we have $|d(x_1,x_2) - d(x_3,x_4)| \geq \delta$. On the other hand, from (3-4), $|d(\tilde{x}_3,\tilde{x}_4) - d(x_3,x_4)| \leq 2r$. Thus, from Triangle inequality:

$$\begin{aligned}|d(\tilde{x}_3,\tilde{x}_4) - d(x_1,x_2)| \geq& |d(x_1,x_2) - d(x_3,x_4)| - |d(\tilde{x}_3,\tilde{x}_4) - d(x_3,x_4)| \\ \geq& \delta - 2r \geq 2r\end{aligned}$$

Since $D_\pi(X,X_r) \geq \frac{1}{2}|d(x_1,x_2) - d(\tilde{x}_3,\tilde{x}_4)|$, we establish $D_\pi(X,X_r) \geq r$ for $\pi \in N_1$.

From the above analysis, $D_\pi(X,X_r) \leq D_\tau(X,X_r)$ for all $\pi \in N_0$ and $\tau \in N_1$. Combining with (3-3), it is clear that the identity permutation is the solution of (3-2), which proves the Lemma. □

With the Lemma, Theorem 3-1 can be proved by choosing a specific projection perturbation such that its discrete Gromov-Hausdorff distance is always bigger than the upper bound for such distance of any orthogonal perturbations. The proof of the Theorem is stated below:

Proof. Applying Lemma 3-1 to the orthogonal perturbation $Y = X_r^\perp$ and projection perturbation $Z = X_r^{\text{Proj}}$, and for any r less than the minimum of the ε specified in Lemma 3-1, we obtain:

$$d_J(X,Y) = \frac{1}{2}\max_{i,j}|d(x_i,x_j) - d(y_i,y_j)| \quad \text{and} \quad d_J(X,Z) = \frac{1}{2}\max_{i,j}|d(x_i,x_j) - d(z_i,z_j)|$$

From the triangle inequality (similar to how we show Eq. (3-4)), we have:

$$\frac{1}{2}\max_{i,j}|d(x_i,x_j)-d(y_i,y_j)|<r \quad \text{and} \quad \frac{1}{2}\max_{i,j}|d(x_i,x_j)-d(z_i,z_j)|\leq r$$

where the first inequality is strict due to the orthogonality of the perturbation.

Given such r, consider the following perturbation $Z = X_r^{\text{Proj}}$ where x_1, x_2, z_1, z_2 are collinear and $|d(x_1,x_2)-d(z_1,z_2)| = 2r$, and $z_i = x_i$ for $i = 3, \ldots, n$. The d_J distance between such Z and X is greater than or equal to r, which proves the theorem. □

This theoretical result suggests the following perturbation scheme: Given a manifold embedded in an affine subspace of $\mathbb{R}^N$ and a fixed amplitude r of perturbation, perturbing the manifold in the orthogonal directions with respect to the affine subspace is preferable to random perturbation. The reason is perturbing along the orthogonal directions minimizes the topological difference between the perturbed manifold and the original.

3.4 Persistent Homology Analysis With the Bottleneck Distance

The results from the previous section show that on the worst-case basis, the orthogonal perturbation is preferable to the projection perturbation. However, when we apply them to actual datasets, how do they compare on average? Since the discrete Gromov-Hausdorff distance is still computationally intractable for a Monte-Carlo analysis, we choose a different approach: *persistent homology*.

For the last 30 years, there have been new developments in the field of algebraic topology, which was classically very abstract and theoretical, toward real-world applications. The newly discovered field, commonly referred to as *applied topology* or *topological data analysis*, is centered around a concept called *persistent homology*. Interested readers can refer to [48, 35, 47] for an overview of the subject.

For any topological space, homology is a topological invariant that counts the number of *holes* or *voids* in the space. Intuitively, the 0th-homology counts the number of connected components, the 1st-homology counts the number of loops, the 2nd-homology counts the number of 2-dimensional voids, and so on. The homology groups of dimension i are denoted by H_i.

Given a point cloud sampled from a manifold, we want to recapture the homological features of the original manifold from these discrete points. The idea is to construct a sequence of topological spaces along some *timeline* and track the evolution of the topological features across time. The longer the features *persist* (and hence the name *persistent homology*), the more likely they are the actual features of the original manifold. Given a point cloud X and a dimension i, the persistence diagram $D_i(X)$ is the set of points $(b,d) \in \mathbb{R}^2$ corresponding to the birth and death time of these features in the aforementioned timeline.

For two point clouds, and in particular for their two persistence diagrams, there are several notions of distances between them, representing how *similar* they are as topological spaces. The most commonly used distance in practice is the *Bottleneck distance*.

Definition 5 (Bottleneck distance)**.** *Let X and Y be two persistence diagrams. The Bottleneck distance $W_\infty(X,Y)$ is given by*

$$W_\infty(X,Y) = \inf_{\varphi: X \to Y} \sup_{x \in X} \|x - \varphi(x)\|_\infty$$

where the infimum is taken over all matchings $\varphi : X \to Y$ (which allows matchings to points with equal birth and death time).

For simplicity, we shorthand the Bottleneck distance between persistence diagrams of X and Y in dimension i to $H_i(X,Y)$ instead of $W_\infty(D_i(X), D_i(Y))$. As the notation takes in 2 parameters in X and Y, this is not to be confused with the homology group of the specified spaces. Note that two point clouds with small Bottleneck distances can be considered topologically similar.

Notably, the bottleneck distance is highly correlated to the Gromov-Hausdorff distance (Section 3.3), as the bottleneck distance of two persistence diagrams of the same dimension is bounded above by their Gromov-Hausdorff distance [22]: $W_\infty(D_i(X), D_i(Y)) \leq d_{GH}(X,Y)$, for every dimension i.

The Bottleneck distance is much more calculable than the Gromov-Hausdorff distance, and there are available software packages depending on the use cases. As such, we run Monte-Carlo simulations to compute the Bottleneck distances on 5 synthetic datasets, 3 real-world tabular datasets, and 2 real-world image datasets to confirm our hypothesis that the orthogonal perturbation preserves the topology better than the projection perturbation *on average*. The synthetic datasets are some noisy point clouds of certain 2-dimensional shapes in 3-dimensional space. The tabular datasets are the COMPAS [62], German Credit [60], and Communities and Crime [110]. The image datasets are MNIST [79] and Fashion-MNIST [148]. Table 3-1 and 3-2 provide more details about those datasets. We use the Ripser Python library [135] to compute the Bottleneck distances in our experiments. All reported Bottleneck distances are normalized with the noise added to the point clouds for more intuitive visualization.

Table 3-3 reports the means H_0 and H_1 Bottleneck distances of the perturbations on the synthetic datasets. The number of data points and the noise level are chosen mainly for nice visualizations (shown in Appendix C). The results show that orthogonal perturbation consistently results in lower H_0 distances for line-shaped datasets and lower H_1 distances for cycle-shaped datasets. Note that in general, H_1 is the better topological indicator for cycle-shaped datasets compared to H_0, since cycles or holes are harder to replicate than connected components.

For the real-world dataset, we conduct the experiments with perturbations of different noise levels and report results in Fig. 3-6 and 3-7. It can be observed that both H_0 and H_1 Bottleneck distances of the persistence diagrams of the orthogonal perturbation are significantly smaller than those of the projection perturbation in all experiments.

3.5 EMaP Algorithm

We now discuss our perturbation-generating algorithm, called EMaP, leveraging the topological information of the input data. To generate better perturbations for the explaining task, EMaP not only exploits the perturbation direction but also uses low-dimensional distances to better capture the notion of locality.

In particular, predictions on the perturbations do not necessarily hold local information about the explained inputs, which is one main reason for the usage of some distance or kernel functions measuring how similar the perturbations are to the explained input. Note that those distances and kernels are normally functions of other distances such as L_1 and L_2 in the input space $\mathbb{R}^N$, which might not correctly capture the notion of similarity. By operating in the low-dimensional manifold, EMaP can overcome those issues. First, if topological similarity implies similarity in the model's predictions, maintaining the topological structure of the original data should improve the relevance of the model's predictions on the perturbations. Therefore, explaining the model with orthogonal perturbations, which helps preserve the topology better, should be more beneficial. Furthermore, the manifold provides a natural way to improve the similarity measurement among data points. Fig. 3-8 shows the issue of similarity measurement based on Euclidean distance in the input space $\mathbb{R}^N$. As the distance ignores the layout of the data, further points on the manifold might result in the same similarity measure. On the other hand, the low-dimensional distances computed on the manifold take into account the layout and can overcome that issue.

Input: Data to explain x_0, a subset of training data (X, y), number of pivots per labels p, number of perturbations per pivot k, lower dimension V and noise level r.
Output: X_r and D_r. X_r contains $k(pl+1)$ orthogonal perturbations locally around x_0 and points in X. D_r contains the low-dimensional distances of points in X_r to x_0 (l is the number of unique labels in y).

```
 1: Initialized an EMaP sampler object $\mathcal{M}$.
 2: $\mathcal{M}$.mapper $\leftarrow$ Mapper to the manifold of dimension $V$ of $X$
 3: $\mathcal{M}$.pivots $\leftarrow \emptyset$
 4: Include $x_0$ to $\mathcal{M}$.pivots
 5: **for** each class $l$ in $y$ **do**
 6:     Include $p$ samples of class $l$ to $\mathcal{M}$.pivots
 7: **end for**
 8: $X_r \leftarrow \emptyset$, $D_r \leftarrow \emptyset$
 9: **for** each data point $x$ in $\mathcal{M}$.pivots **do**
10:     $\tilde{X}, \tilde{D} \leftarrow \mathcal{M}$.gen_perturbation$(x, k, r)$
11:     Include $\tilde{X}$ to $X_r$ and include $\tilde{D}$ to $D_r$
12: **end for**
13: **return** $X_r$ and $D_r$.
```

Algorithm 3-1. EMaP algorithm

3.5.1 Algorithm Overview

Given an input to be explained, the output of EMaP are the perturbations along the manifold's orthogonal directions X_r and their low-dimensional distances D_r to that input. The pseudo-code of EMaP is shown in Alg. 3-1. The first step is to learn an embedding function, i.e. a *mapper*, transforming the data to the low dimension (line 2). Then, p samples from each label are selected and combined with the explained input x_0 into a set, called the *pivots* (lines 3 to 7). After that, EMaP generates perturbations along the orthogonal directions of the manifold from each pivot (line 10). The usage of pivots is to provide the explainer a wider range of perturbations for better performance.

3.5.2 The Mapper

In EMaP, the mapper is learned from a manifold approximated by UMAP [90]. Since the manifold learned by UMAP is optimized for global information, the orthogonal directions computed on top of that manifold at local data points are prone to high error. This can degrade the correctness of orthogonal perturbations significantly. To overcome this issue, EMaP learns a local-subspace for each pivot and generates the orthogonal perturbations on top of that subspace. Intuitively, the local-subspace is a local affine approximation of the manifold. Therefore, the resulting orthogonal directions are more finely tuned for the local points. Let G_x denote the $N \times V$ matrix characterizing the local-subspace at x. Since G_x is a linear approximation of data points near x, by denoting $\omega : \mathbb{R}^N \to \mathbb{R}^V$ as the function embedding input data to the manifold, we have $z^{\text{low}} := \omega(z) \approx G_x^\top z$ and $z \approx G_x \, z^{\text{low}}$, where $z \in \mathbb{R}^N$ are points near x and z^{low} are their embedding in $\mathbb{R}^V$. In the current implementations, the set of pivots contains the explained data point and p data points sampled from each class label l (see Algorithm 3-1).

3.5.3 EMaP Orthogonal Perturbations

The key step of EMaP is the generation of orthogonal perturbations $\tilde{x}$ from x (line 10, Alg. 3-1), which can be expressed as:

$$\tilde{x} = x + \text{noise} - \text{Proj}_{G_x}(\text{noise}) \tag{3-5}$$

```
Input: Input $x$, number of perturbation $k$ and noise level $r$.
Output: $k$ orthogonal perturbations of $x$ and their low-dimension distances to $x$.
 1: $G_x \leftarrow$ self.get_local_subspace($x$)
 2: $\tilde{X} \leftarrow \emptyset$
 3: for $1 \leq i \leq k$ do
 4:     noise $\leftarrow \mathcal{N}(0, \Sigma_r)$
 5:     $\tilde{x} \leftarrow x + \text{noise} - \text{Proj}_{G_x}(\text{noise})$
 6:     Include $\tilde{x}$ into $\tilde{X}$
 7: end for
 8: $\tilde{X}^{\text{low}} \leftarrow$ self.mapper.transform($\tilde{X}$)
 9: $x^{\text{low}} \leftarrow$ self.mapper.transform($x$)
10: $\tilde{D} \leftarrow$ distances of each member in $\tilde{X}^{\text{low}}$ to $x^{\text{low}}$
11: return $\tilde{X}$ and $\tilde{D}$
```

Algorithm 3-2. gen_perturbation

where the noise is sampled from a multivariate normal distribution and $\text{Proj}_{G_x}(\text{noise})$ is the projection of the noise on the local-subspace characterized by G_x. Upon obtaining the orthogonal perturbations, EMaP can compute their low-dimensional embedding using the mapper transform function ω. The pseudocode for this computation is in Algorithm 3-2.

3.5.4 Local-Subspace Approximation

The correctness of the orthogonal perturbations, i.e. whether the perturbations are lying in the orthogonal subspace, is dependent on the correctness of the computation of G_x. Ideally, given a set of data Z near x in the manifold, we solve the following optimization problem for G_x :

$$G_x = \arg\min_{G} \sum_{z \in Z} \|G\omega(z) - z\|, \tag{3-6}$$

where ω is the transform function embedding input data to the manifold learned in the previous step. An intuition is, for all z in the manifold and near x, we are searching for a matrix G such that the inverse mapping $G\omega(z) \approx Gz^{\text{low}}$ approximately equals its original value z. Note that if the embedding ω is exactly on Z and the manifold is affine, the optimization (3-6) can achieve its optimal value 0 for some G. Since it is not trivial to obtain the set Z belonging to the manifold, EMaP perturbs around x with some random noise and solves the following optimization instead:

$$\hat{G}_x = \arg\min_{G} \sum_{r \in B} \|G\omega(x+r) - (x+r)\|, \tag{3-7}$$

where B is a ball centering at 0 with noise radius r. EMaP solves (3-7) for an approximation of the local-subspace. Further details of this step are provided in Algorithm 3-3

```
**Input**: Data point $z$.
**Hyper-parameters**: Number of training samples $k_T$ and noise level for training $r_T$.
**Output**: Matrix $G$ characterize the local-subspace at $z$
1: $\tilde{Z} \leftarrow \emptyset$
2: for $1 \leq i \leq k_T$ do
3:     noise $\leftarrow \mathcal{N}(0, \Sigma_{r_T})$
4:     $\tilde{z} \leftarrow z + $noise
5:     Include $\tilde{z}$ into $\tilde{Z}$
6: end for
7: $\tilde{Z}^{\text{low}} \leftarrow$ self.mapper.transform$(\tilde{Z})$
8: $G \leftarrow \arg\min_W \|\tilde{Z} - W\tilde{Z}^{\text{low}}\|_2$
9: return $G$
```

Algorithm 3-3. get_local_subspace

The gap between the ideal solution G_x (3-6) and the approximation $\hat{G}_x$ (3-7) can be characterized by the error between $\{G_x\omega(z)\}$ and $\{\hat{G}_x\omega(z)\}$. Lemma 3-2 provides a bound on that reconstruction error where the set Z in (3-6) is the projection of a ball B on the data manifold. The bound holds under a mild assumption that the optimal objective of (3-6) is not larger than that of (3-7). Since Z is in a subspace of dimension V and the set of $x+r$ is a ball in $\mathbb{R}^N$, finding a subspace of dimension V approximating the subspace containing Z should give a lower error.

Lemma 3-2. *Assume that all data points x belong to the same affine space $\mathbb{R}^V$. Let Proj be the projection onto $\mathbb{R}^V$, then under the above assumption on the optimization (3-6) and (3-7), the reconstruction error on perturbed data points is upper bounded by:*

$$\|\hat{G}_x\omega(x+r) - G_x\omega(\text{Proj}(x+r))\| \leq F_B(\omega) + \|r^{\perp}\|,$$

where $r^{\perp}$ is the orthogonal components of r and $F_B(\omega) := \min_G \sum_{r \in B} \|G\omega(x+r) - (x+r)\|$.

Proof. The assumption regarding the objectives (3-6) and (3-7) mentioned in the Lemma can be rewritten as:

$$\sum_{r \in B} \|G_2 \omega(\text{Proj}(x+r)) - \text{Proj}(x+r)\| \leq \sum_{r \in B} \|G_1 \omega(x+r) - (x+r)\|$$

With that, we have:

$$\begin{aligned}
&2\|G_1 \omega(x+r) - (x+r)\| \\
\geq &\|G_1 \omega(x+r) - (x+r)\| + \|G_2 \omega(\text{Proj}(x+r)) - \text{Proj}(x+r)\| \\
\geq &\|G_1 \omega(x+r) - G_2 \omega(\text{Proj}(x+r)) - (x+r-\text{Proj}(x+r))\| \\
\\
\geq &\|G_1 \omega(x+r) - G_2 \omega(\text{Proj}(x+r))\| - \|(x+r) - \text{Proj}(x+r))\| \\
= &\|G_1 \omega(x+r) - G_2 \omega(\text{Proj}(x+r))\| - \|r - \text{Proj}(r)\|,
\end{aligned}$$

where the last two inequalities are from the Triangle Inequality. The last equality is due to the fact that $(x+r) - \text{Proj}(x+r)$ is the orthogonal components of $x+r$ and that x has no orthogonal components. □

Note that $F_B(\omega)$ is small if the manifold is affine in the neighborhood B of x and ω is good, i.e. $\hat{G}_x$ is a good estimator for G_x under the above assumption.

3.6 Experiments

EMaP is heuristically evaluated on two main objectives: the explainer's performance and the perturbations' robustness. The experiments are conducted on 3 tabular datasets, 2 image datasets, and 4 text datasets of reviews in multiple domains. They are COMPAS [62], Communities and Crime [110], German Credit [60], MNIST [79], Fashion-MNIST [148], and 4 reviews datasets in the Multi-Domain Sentiment [15].

3.6.1 Experimental Settings

We first describe how the datasets and the testing models are processed in our experiments. Then, we provide some details of the implementation of EMaP along with the baselines.

Regarding the datasets and the models' hyperparameters, all reported results for real-world datasets include at least 2000 data points and 100 runs, except for those of German Credit where the data only consists of 1000 samples. The text datasets are tokenized into vectors of 1000 input features while the MNIST and Fashion-MNIST datasets are of dimension $1 \times 28 \times 28$ and $3 \times 32 \times 32$, respectively. The model's inputs of all experiments are normalized between 0 and 1. The experimental model for the text dataset is the $L1$ logistic regression as used by LIME [111]. The models of testing for the two image datasets are 2-layer convolutional networks implemented in Pytorch [101] with test set accuracy of 98% for MNIST and 93% for Fashion-MNIST.

We now discuss the technical implementation of EMaP and the baselines. Generally, EMaP perturbations can be used to leverage any model-agnostic perturbation-based methods; however, the modification may require significant changes on the existing implementations of the methods. In the current experiments, we use EMaP to leverage LIME [111] as a proof of work. LIME is chosen to demonstrate the usage of EMaP since it requires few modifications to exploit EMaP's perturbations. This helps demonstrate fairly the gain of applying EMaP on explanation methods. The notation EMaP-LIME (or EMaP for short) is used to indicate *LIME with EMaP's perturbations* in the reported following experimental results.

To apply EMaP into LIME, we simply choose the loss function (defined in Eq.(1-1)) as:

$$\mathcal{L}(f, g, \pi_x) = \sum_{\tilde{x} \in X_r} \pi_x(\tilde{x})(f(\tilde{x}) - g(\tilde{x}))$$

where $\pi_x(\tilde{x}) = \exp(-\tilde{D}(x, \tilde{x})^2/\sigma^2)$ with the distance $\tilde{D}$ computed as in Algorithm 3-2 and g is the linear function of the features of the perturbation.

Besides LIME, EMaP-LIME is also compared with some other black-box and white-box explanation methods. Following are the brief descriptions of those methods:

1. LIME zero: LIME with perturbations whose perturbed features are set to zero. This method is used by LIME in explaining text data.

2. LIME+: LIME with perturbations whose perturbed features are added with Gaussian noise. Ths method is used by LIME in explaining image data.

3. LIME*: LIME with perturbed whose perturbed features are multiplied with uniform noise between 0 and 1. This can be considered as a smoother version of LIME zero. We include this variant of LIME because it provides nice explanations in some specific experiments (see Appendix. C for some examples).

4. KernelSHAP: a black-box method based on Shapley value [86], whose perturbed features are set to the average of some background data.

5. GradientSHAP: a white-box method based on Shapley value [86], which relies on the gradient of the model.

6. DeepLIFT: a white-box method based on back-propagating the model [122].

7. Parzen [10]: an explanation method that approximates the model globally and uses the gradients of that approximating model to explain the predictions.

8. Greedy: the features contributing the most to the predicted class are removed and selected until the prediction changes. The implementation is taken from [111].

The noise vector used to generate perturbations for all applicable explainers has a radius 10^{-3} for text data and 10^{-4} for image data, which are in the range analyzed in the previous Bottleneck distance experiments in Fig. 3-6 and 3-7. The noise radius r_T used to approximate the local-subspaces of EMaP (Algorithm 3-3) is chosen equal to the noise radius r for perturbation. n_components $\in \{2,3\}$ and min_dist $= 0.1$ (default value) are chosen for UMAP's hyper-parameters. Finally, for fair comparison, the number of perturbations used to generate the explanations of all reported methods are 1000.

3.6.2 Explaining Performance of Explanation Methods

For illustrative purposes, Fig. 3-9 shows the resulting explanations on MNIST and Fashion-MNIST generated by EMaP and some other black-box methods (More examples are in Appendix C). All methods use 1000 perturbations to generate the corresponding explanations. We can see that, while EMaP and SHAP capture relevant features, LIME includes a lot of noisy features in the background. Note that the reason LIME* does not include the background is that the background of the original image has a value of 0 and, through multiplication, the features are not perturbed. This makes their weights 0 in the explaining model g. Regarding the low quality of LIME zero and LIME+, the main reason is that the method would require a much larger number of perturbations to generate good results at the pixel level. We want to point out that the only difference between EMaP and LIME is in the perturbations and how to compute the distances between the perturbations and the original data points. This clearly demonstrates optimizing the perturbations can greatly improve the explanation's quality. Another interesting observation is the red areas on top of the digit 4 and at the bottom of the *Trouser* (the second and third rows of Fig. 3-9) only identified by EMaP. In fact, those areas are essential to the predictions since they differentiate the samples from other classes, i.e., digit 9 and the *Dress*, respectively. Those areas, on the other hand, are neglected by all other examples.

For the sentiment classification task, we use the Multi-Domain Sentiment dataset [15]. We follow the experimental setup in [111], in which the ground-truth explanatory features of the $L1$ logistic regression model are known. In particular, a regression model is trained for each dataset and the ground-truth explanatory features are non-zero coefficients. For each explanatory budget, i.e., the number of features included in the explanations, the fraction of these ground-truth features recovered by the explanations is computed. The performance of an explainer is evaluated by the precision and the recall rate of the features in the explanations. Intuitively, the more features included in the explanation, the higher the recall and the lower the precision.

Fig. 3-10 shows the scatter plot of precision vs. recall of LIME and LIME with EMaP on 4 review datasets of *books*, *dvds*, *kitchen* and *electronics*. Similar to how LIME is evaluated in these datasets in its original paper, we compare LIME with EMAP to LIME, Greedy and Parzen explanation methods [10]. In Greedy, the features contributing the most to the predicted class are removed until the prediction changes. On the other hand, Parzen approximates the model globally with Parzen windows and the explanation is the gradient of the prediction. The results clearly show that EMaP consistently improves the faithfulness of LIME.

Since there are no ground-truth explanations for the image datasets, we evaluate explanations using the RDT-fidelity [44], the infidelity scores [152] and the log-odds scores [122]. The RDT-fidelity of an explanation S with respect to a model f is the probability that the model's prediction does not change if the features in the explanation remain unchanged:

$$RDT(S) = \mathbb{E}\left[\mathbf{1}\left(f(x) = f(x_S)\right)\right],$$

where x_S is x with noise added on features not in S and the expectation is taken over the noise. On the other hand, the infidelity score measures the expected error between the explanation multiplied by a meaningful perturbation and the differences between the predictions at its input and at the perturbation. The metric can be considered as a generalized notion of Sensitivity-n [6]. Intuitively, explanations with lower infidelity are more desirable. Finally, given the importance weights on features (i.e., the explanation), the log-odds score measures the difference between the image and the modified image whose pixels are erased based on their importance weights. Intuitively, the higher the log-odds score, the better the explanation.

The evaluations using the RDT-fidelity are computed on explanations of $K = 40$ and 80 features of the explained image (Fig. 3-11). The results clearly demonstrate the high fidelity of our EMaP in both image datasets. Fig. 3-12 shows the infidelity scores. It is clear that EMaP has the lowest infidelity score among all black-box methods. Even though the white box methods, KernelSHAP and DeepLIFT, have more information on the explained models than EMaP, they can only outperform EMaP in Fashion-MNIST. Additionally, Fig. 3-13 shows the log-odds scores of EMaP with low-dimension $d = 2$ and $d = 3$, along with other explanation methods and other perturbation schemes. In these experiments, the scores are computed on 20% erased pixels. In MNIST, we can see that EMaP does not degrade the explainer performance compared to LIME in terms of log-odds (note that the default setting for LIME for image data is LIME+). For Fashion-MNIST, EMaP improves the log-odds significantly. More examples of EMaP and other explanation methods are provided in Appendix C.

3.6.3 Perturbation's Robustness

The robustness of the perturbations is evaluated based on the discriminator's performance in differentiating perturbations from the original data. Following the setup in [125], the discriminator is trained with full knowledge of the explainer's parameters. This discriminator has been shown to be able to recognize perturbations of LIME and SHAP explainers. As DeepLIFT and GradientSHAP are white-box methods and do not use perturbations, the evaluation if perturbation's robustness is only applicable to LIME and KernelSHAP. Since there is no ambiguity, SHAP is used to denote KernelSHAP in the following results.

The experimental results show that EMaP's perturbation is more robust to the discriminator. Specifically, Figs. 3-14, 3-15, 3-16 and 3-17 show the True-Positive (TP) and True-Negative (TN) rates of discriminators on perturbations of 2 image datasets and 2 tabular datasets. For image datasets, the discriminators can easily recognize perturbations generated by LIME and SHAP. On the other hand, EMaP perturbations significantly lower the success rates of discriminators in recognizing the perturbations. While the explainers' perturbation schemes show to be slightly more robust in the tabular datasets compared to the image datasets, EMaP still improves the perturbations' robustness remarkably.

3.6.4 Computational Resource and Run Time

The reported experiments are conducted on a single GPU-assisted compute node that is installed with a Linux 64-bit operating system. The allocated resources include 32 CPU cores (AMD EPYC 7742 model) with 2 threads per core and 100GB of RAM. The node is also equipped with 8 GPUs (NVIDIA DGX A100 SuperPod model), with 80GB of memory per GPU.

The run time of EMaP is mostly dictated by the learning of the embedding function. That initialization step in the tabular dataset for about 2000 data points takes less than 2 minutes. It takes between 240 and 260 seconds for all images of 60000 MNIST/Fashion-MNIST images. Note that the manifold and local-subspaces can be computed before deployment since it does not depends on the explained inputs. Thus, this overhead of EMaP can be mitigated at deployment. Table 3-4 reports the actual run time of EMaP and LIME in the image datasets.

3.7 Conclusion, Limitations and Future Research

From the theoretical and experimental results, we propose the EMaP algorithm to exploit the data manifold to preserve the topology information in its perturbation. We recognize the main limitation of EMaP is in its requirement of low-dimensional representations of the data and the local affine subspaces. For more complex data, computing them correctly can be very challenging. There are several interesting open questions of EMaP for future work. For instance, it is important to study the impact of the underlying manifold-learning algorithm, i.e. the UMAP, on the perturbations and the explanations. It is also interesting to examine the behavior of EMaP in a wider range of explainers and applications.

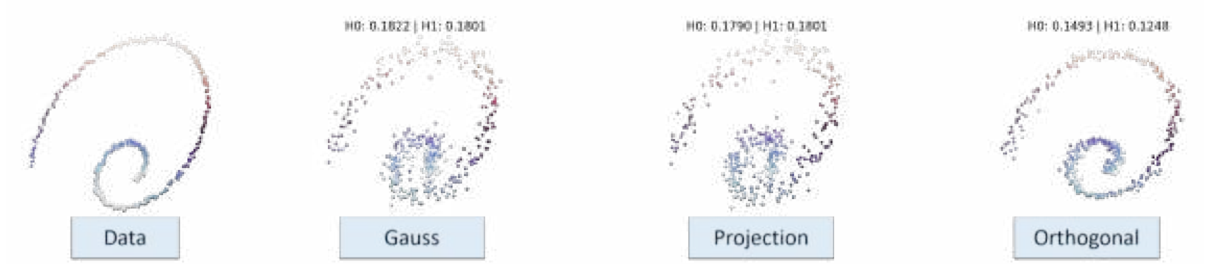


Figure 3-1. Visualization of perturbations with the same magnitude generated from a point cloud of a 2-dimensional spiral. Perturbations along the orthogonal directions of the data subspace (far-right) result in lower topological distortion, i.e. smaller Bottleneck distances H_0 and H_1 (the Bottleneck distance is discussed in Section 3.4).

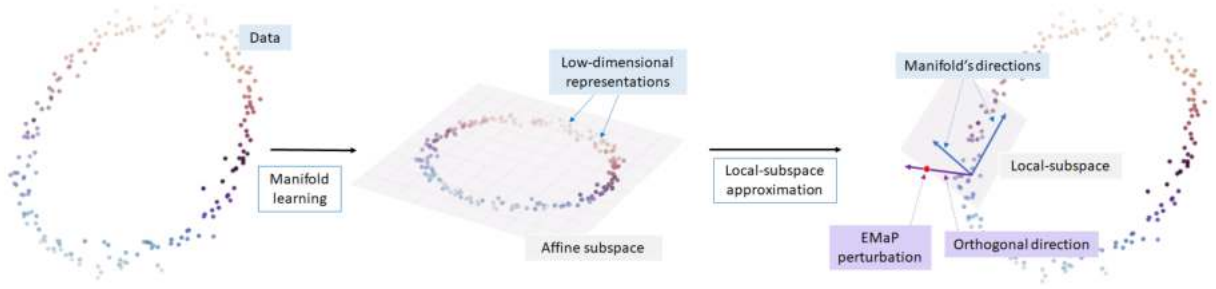
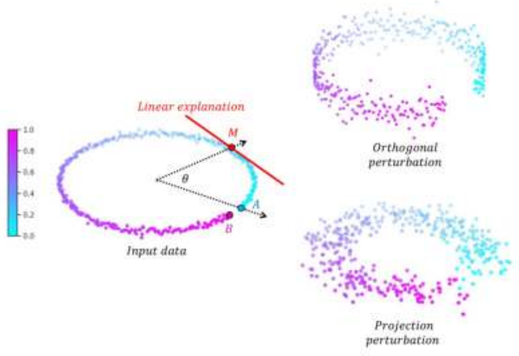
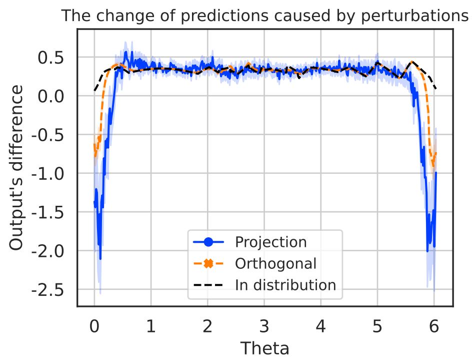


Figure 3-2. EMaP's perturbation: Assume the data is embedded in a low-dimensional affine subspace (middle figure), EMaP approximates that subspace locally at some given data points and performs perturbation along orthogonal directions of that subspace (right figure).

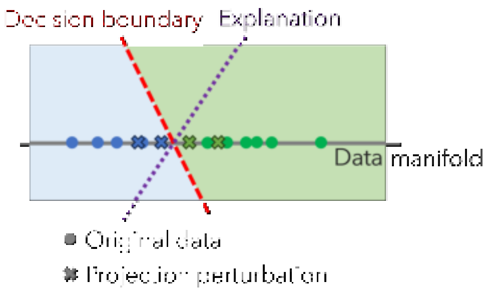


(a) A C-shaped data in a regression task (left) and its perturbations (right). The projection perturbation of the data is more likely to destroy the *gap* at the two ends of the original C-shaped data.

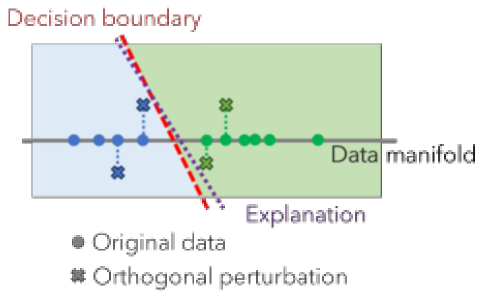


(b) The figure reports the normalized change $\frac{|f(x+\delta)-f(x)|}{\|\delta\|_{L_2}}$ for x along the C-shaped data (located by the angle θ). The perturbation $x+\delta$ belongs to the projection point cloud, the orthogonal point cloud, and the original data as indicated in the legend.

Figure 3-3. Example showing how leveraging topological information can be beneficial for the explanation task.



(a) Explaining using projecting perturbation.



(b) Explaining using orthogonal perturbation.

Figure 3-4. A 1-D example showing the impact of perturbation's direction.

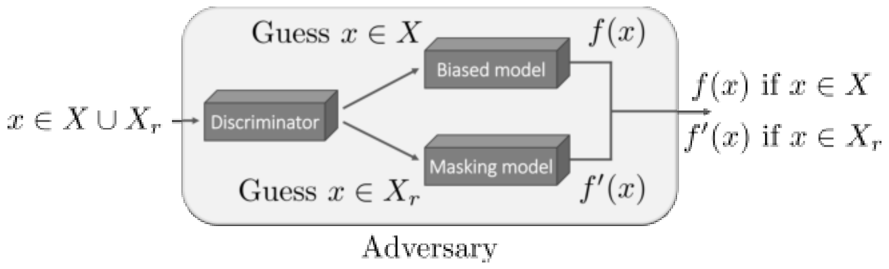


Figure 3-5. The discriminator-based attack framework: By recognizing and forwarding the perturbations X_r generated by an explainer to the masking model f', the biased-model f can be deployed without detection.

Table 3-1. The parameters of the Bottleneck distance's experiments on the synthetic datasets. The perturbation column shows the average magnitude of the perturbation on the data.

Dataset	Parameter	No. points	Data's noise	Perturbation	No. runs	EMaP's dim
Line	Length: 10	100	0.1	≈ 0.15	100	1
Circle	Radius: 1	400	0.1	≈ 0.1	100	2
2 intersecting circles	Radius: 1	400	0.01	≈ 0.1	100	2
2 concentric circles	Radius: 1	400	0.01	≈ 0.1	100	2
Spiral	Radius: [0,2]	1000	0.02	≈ 0.05	100	2

Table 3-2. The parameters of the Bottleneck distance's experiments on the real-world datasets.

Experiment	No. data points	No. feats	Feature's values	No. runs	EMaP's dim
COMPAS	7214	100	{0,1}	100	2
German Credit	1000	28	{0,1}	100	2
CC	2215	100	{0,1}	100	2
MNIST	60000	28×28	[0,1]	100	2 and 3
Fashion-MNIST	60000	28×28	[0,1]	100	2 and 3

Table 3-3. The normalized H_0 and H_1 Bottleneck distances for Gaussian (G), projection (P), and orthogonal (O) perturbations on synthetic datasets. Visualizations of the actual perturbations are provided in Appendix C.

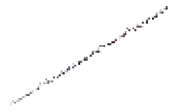
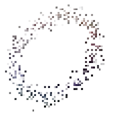
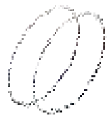
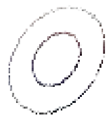
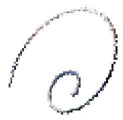
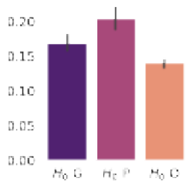
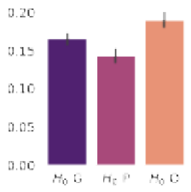
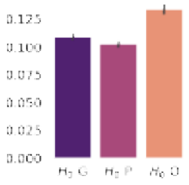
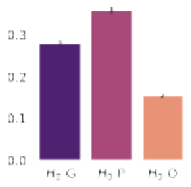
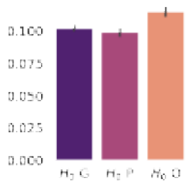
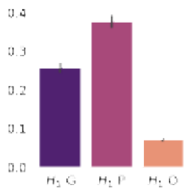
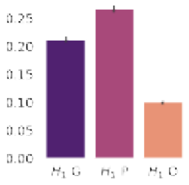
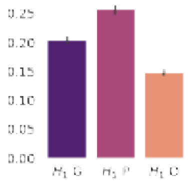
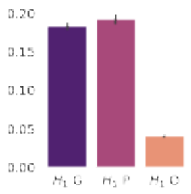
Data					
H_0					
H_1					

Table 3-4. Run time (in seconds) of EMaP and LIME in generating explanations using 1000 perturbations per explanation. The reported numbers are *seconds* (for initialization column) and *seconds per explanation* (for other columns).

	EMaP Initialization	LIME	EMaP (d=2)	EMaP (d=3)
MNIST	240-260	0.763	1.311	1.493
Fashion-MNIST	240-260	0.726	1.502	1.467

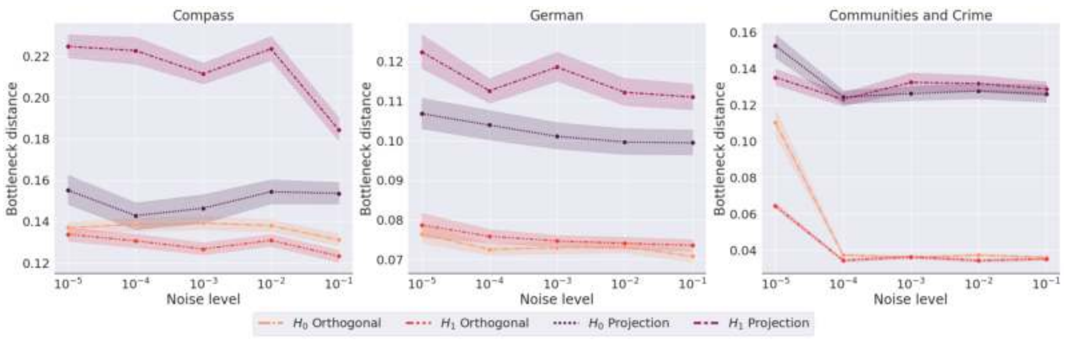


Figure 3-6. The normalized H_0 and H_1 Bottleneck distances for orthogonal and projection perturbations on 3 real-world datasets at different noise levels. The x-axis shows the average perturbation radius applied on each data point (log-scale).

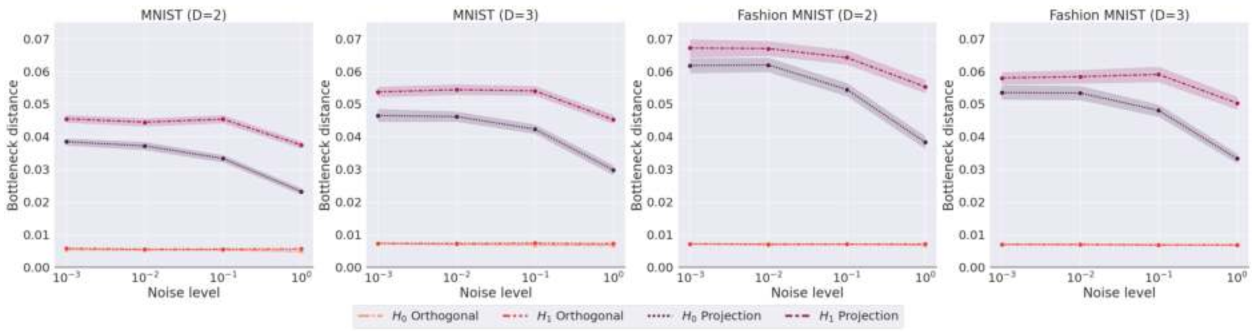


Figure 3-7. The normalized H_0 and H_1 Bottleneck distances for orthogonal and projection perturbations on 2 image datasets at different noise levels. The x-axis shows the average perturbation's radius applied on each data point (log-scale).

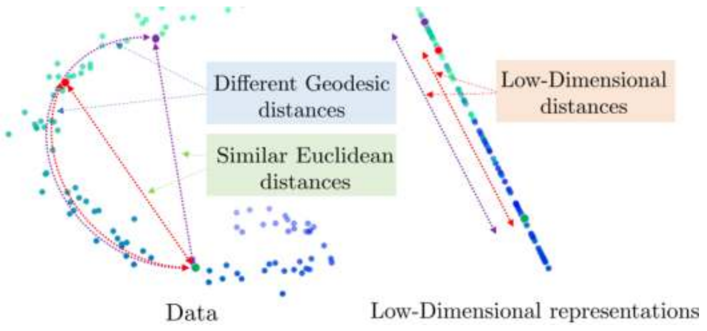

Different Geodesic distances
Similar Euclidean distances
Low-Dimensional distances
Data
Low-Dimensional representations

Figure 3-8. Euclidean distances computing in the input space might not capture the actual distances between the data points (left). Distances in low-dimensional space can help with the issue (right).

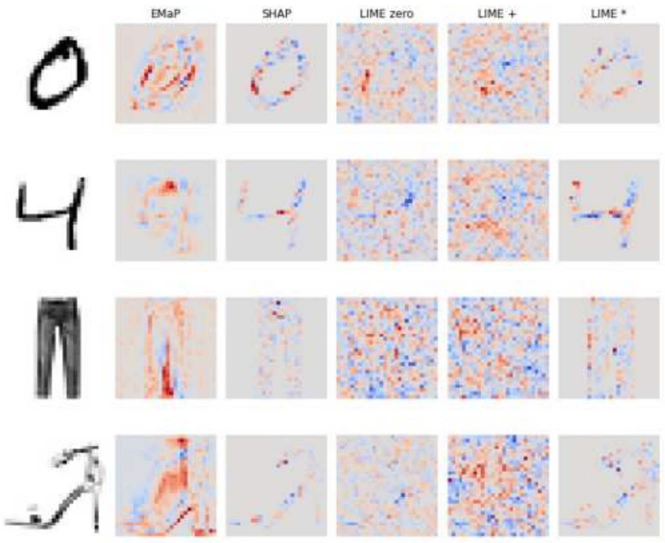


Figure 3-9. Examples of explanations of different methods in MNIST and Fashion-MNIST. Modifying the red-est (blue-est) area would negate (strengthen) the original prediction.

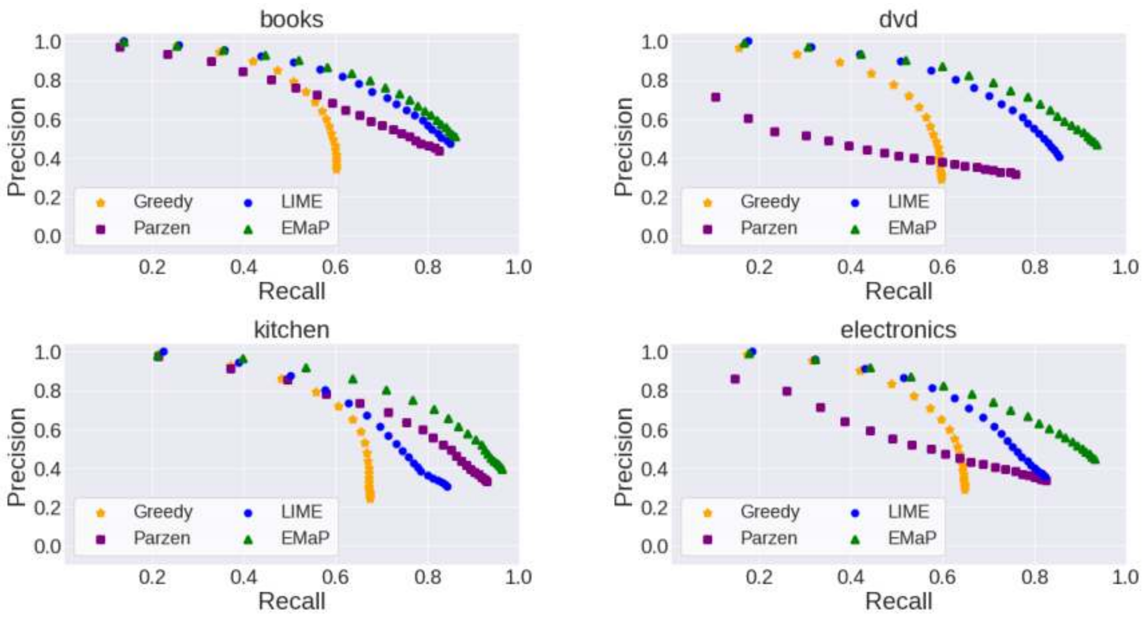


Figure 3-10. The precision and recall of explanations returned by Greedy, LIME, Parzen, and LIME-EMaP (the higher the better). The dots are in the increasing order of the number of features in explanations (left to right).

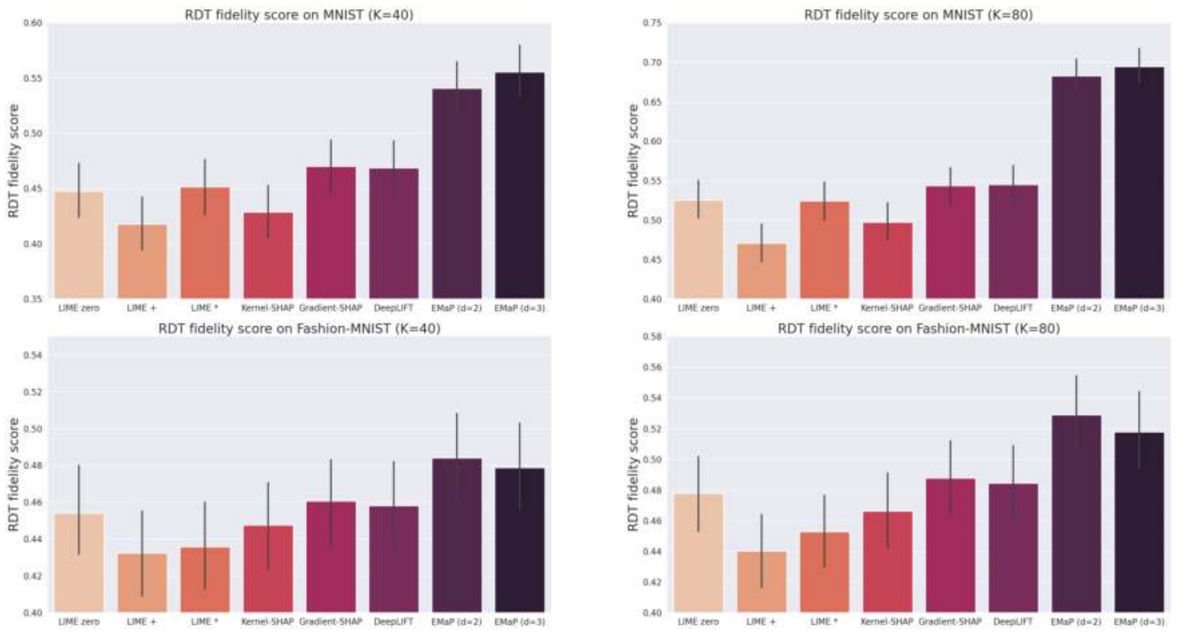


Figure 3-11. RDT fidelity scores of different perturbation-based methods on MNIST and Fashion-MNIST (the higher the better).

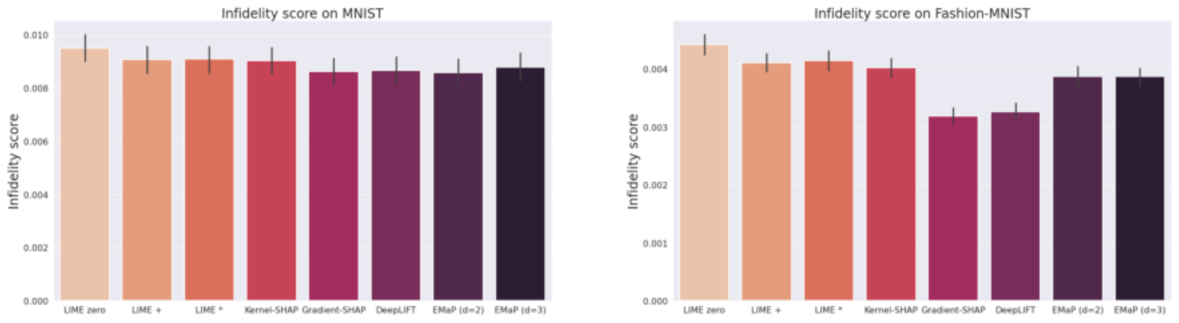


Figure 3-12. Infidelity scores of different perturbation-based methods on MNIST and Fashion-MNIST (the lower the better).

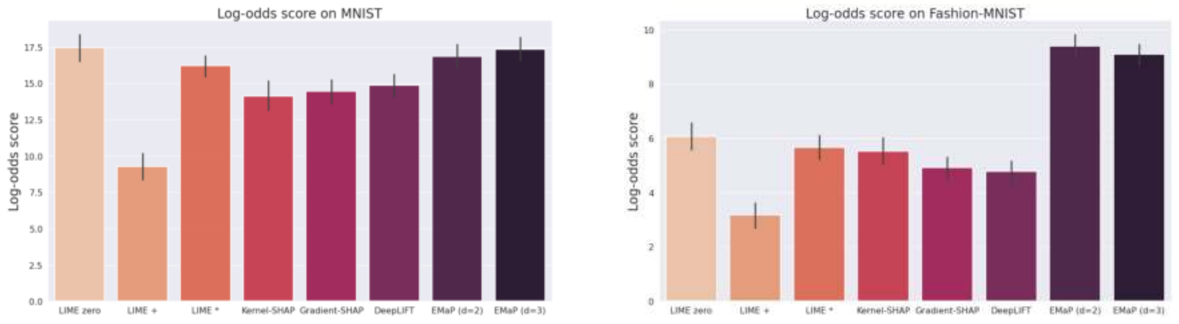


Figure 3-13. Log-odds scores of different perturbation-based methods on MNIST and Fashion-MNIST (the higher the better).

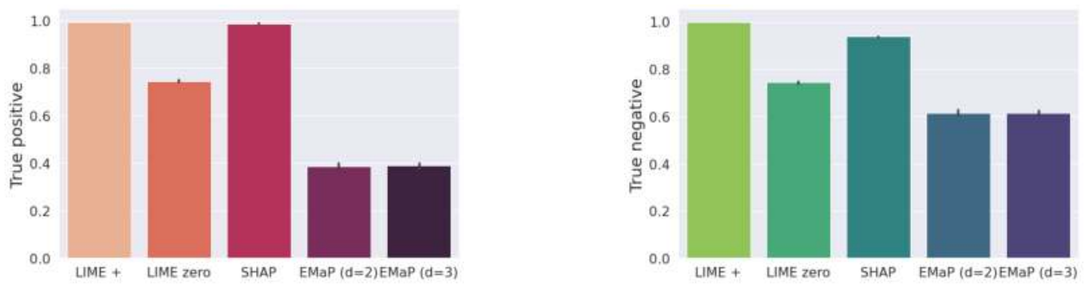


Figure 3-14. True-Positive and True-Negative rates of the discriminators on perturbations of different methods on MNIST dataset (the lower the better).

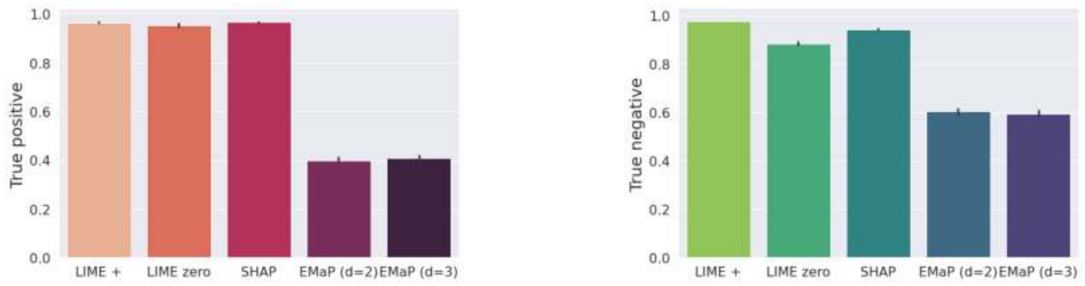


Figure 3-15. True-Positive and True-Negative rates of the discriminators on perturbations of different methods on Fashion-MNIST dataset (the lower the better).

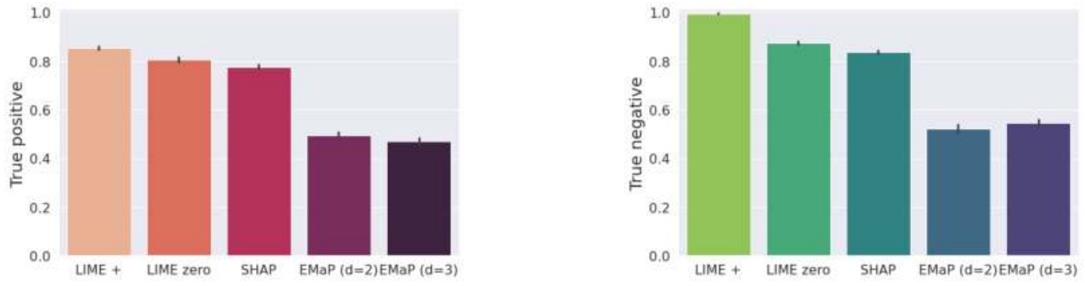


Figure 3-16. True-Positive and True-Negative rates of the discriminators on perturbations of different methods on Communities and Crime dataset (the lower the better).

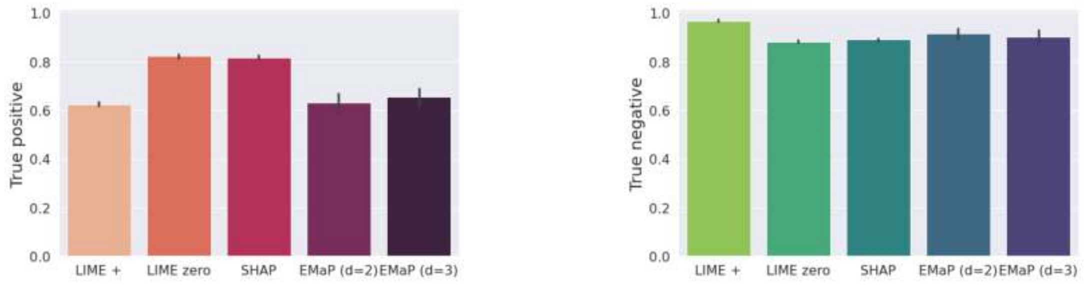


Figure 3-17. True-Positive and True-Negative rates of the discriminators on perturbations of different methods on German Credit dataset (the lower the better).

CHAPTER 4
NEUCEPT: LEARN NEURAL NETWORKS' MECHANISM VIA CRITICAL NEURONS WITH PRECISION GUARANTEE

Significant efforts have been dedicated to improving the interpretability of modern neural networks, leading to several advancements [83, 95]; however, few works have been conducted to characterize local predictions of the neural networks based on the internal forwarding computation of the model. In this chapter, we focus on investigating different mechanisms learned by the neural networks to generate predictions. Intuitively, the mechanism of a prediction is the forwarding process producing the prediction in the examined model (see definition in Sect. 4.2). The assumption of this work is that predictions of the same class label can be generated by different mechanisms which can be captured and characterized by activation of some specific neurons, called *critical neurons*. Analyzing the activation of those neurons can help identify the model's mechanisms and shed light on how the model works.

The following are key reasons motivating this study. First, the identification of critical neurons can serve as an initial model's examination for further study of the model's dynamics. Second, critical neurons allow us to characterize the model's predictions based on how they are generated by the model. Each set of similar predictions can be studied and analyzed for downstream tasks such as performance [38] and trust evaluation [145]. Finally, identifying critical neurons provides a new dimension on how we explain the predictions compared to local attribution explanation methods. This chapter will provide some case studies related to these motivations in Sect. 4.4. For now, Fig. 4-2 provides a concrete example motivating this study. The example considers 2 LeNet classifying *even* or *odd* digits on the MNIST dataset. While the outputs and local explanations hardly show any differences between the two models, the evidence-based on some specific neurons suggests otherwise. This example shows how an explanation at neuron-level can be beneficial.

This chapter is about NeuCEPT - a method to learn neural network's mechanism via critical neurons with precision guarantee (Fig. 4-3). The innovation of NeuCEPT lies in its two main components: NeuCEPT-discovery and NeuCEPT-learning. In NeuCEPT-discovery (Sect. 4.1), we introduce a layer-by-layer mutual information objective to discover the *critical neurons* of the model's predictions. Intuitively, the critical neurons of a layer are the neurons determining the activation of the critical neurons in the sub-sequence layer, which eventually determines the predictions. To solve for these sets of critical neurons efficiently, We develop a pairwise-parallel approximation algorithm with a theoretical precision guarantee. In NeuCEPT-learning (Sect. 4.2), we provide information-theoretic interpretation of the critical neurons and elaborate how learning the mechanism on top of critical neurons can result in better claims on DNNs' mechanisms in terms of explainability power and non-redundancy. We propose an unsupervised learning algorithm, called NeuCEPT-learning, to carry out that task. Additionally, we propose a new testing approach, the prior-knowledge training (Sect. 4.3), to experimentally evaluate the claims on DNNs' predicting mechanism. Rigorous experiments on MNIST and CIFAR-10 show that NeuCEPT consistently detects the embedded mechanisms in the models. Ablation study is also conducted extensively. Finally, this work demonstrates the advantages of NeuCEPT and the identification of critical neurons in some practical tasks (Sect. 4.4), including the study of the model's linear separability [2], the discovery of some interesting neurons and behaviors of predictions of Inception-v3 [132], and the identifications of unreliable predictions in CheXNet [108].

4.1 Critical Neurons Identification with NeuCEPT-Discovery

Although modern DNNs contain thousands to millions of neurons, only a small portion of neurons contributes mostly to the predictions [42, 12]. For now, such neurons are called *critical neurons of the predictions* or *critical neurons* for short. Identifying critical neurons not only reduces the complexity of the mechanism's discovery (Sect. 4.2.2) but also offers more compact explanations for the predictions. Unfortunately, due to the sequential structure of DNNs, identifying critical neurons is a daunting task, from formulating a suitable objective function to solving the problem and interpreting those neurons' activation.

4.1.1 Problem Formulation

Similar to previous chapters, this study considers the examined neural network as a forwarding function $y = f(x)$, where $y \in \mathbb{R}^m$ is a logit and $x \in \mathbb{R}^n$ is an input. The neural network has a sequential structure with L layers, each layer has k_l neurons $(l = 1, ..., L)$. The activation of neurons at any layers on a given input can be computed by forwarding the model. We denote this computation as $z_l = f_l(x)$ where $f_l : \mathbb{R}^n \to \mathbb{R}^{k_l}$. Then $z = [z_0, ..., z_L]$ is the activation of all model's neurons on that input.

The capital letters are used to refer to random variables, i.e. Z_l is the random variable representing the activation of neurons at layer l. The superscript notation refers to the random variables associated with a subset of neurons. For instance, given a subset $\mathcal{S}$ of neurons at layer l, $Z_l^{\mathcal{S}}$ is the random activation of the neurons in $\mathcal{S}$ at layer l. Due to the forwarding structure of the model, the activation of neurons at a given layer depends only on the activation of neurons at the previous layer, i.e. $Z_l \perp\!\!\!\perp Z_j | Z_{l-1}, \forall j = 0, ..., l-2$, where $\perp\!\!\!\perp$ denotes the independent relationship. Thus, we have the Markov chain:

$$X = Z_0 \to Z_1 \to \cdots \to Z_L. \tag{4-1}$$

Given a prediction, a layer l, and the corresponding random activation Z_l, the goal to identify a subset of the critical neurons at that layer, i.e. the subset of neurons containing the most information on the prediction of interest. From an information-theoretic approach, the notion of criticality is formalized using mutual-information (MI) via the critical neuron identification (CNI) problem:

$$\mathcal{S}_l = \text{argmax}_{\mathcal{S} \subseteq \mathcal{N}_l} I\left(Z_l^{\mathcal{S}}; Z_{l+1}^{\mathcal{S}_{l+1}}\right), \text{ s.t. } \mathcal{S} \in \mathcal{C}, \tag{4-2}$$

where $\mathcal{N}_l$ is the set of neuron at layer l, I is the joint mutual-information function [24] and $\mathcal{S} \in \mathcal{C}$ represents some complexity constraints for compact solutions.

Intuitively, at each layer, the CNI searches for the neurons holding the most information on the neurons solved in the next layer. By bounding the first optimization at the last layer L to maximize $I\left(Z_{L-1}^{\mathcal{S}};Y\right)$, where $Y = Z_L^{\{o\}}$ and o is the neuron associated with the prediction's class, we enforce the sub-sequence optimizations at the earlier layers to search for the neurons holding the most information on the examined prediction.

4.1.2 Solutions with Precision Guarantee

As CNI is in NP-hard (In fact, the CNI can be considered as a general version of the feature-selection problem with mutual-information objective, which is known to be NP-hard [66]), we introduce NeuCEPT-discovery to approximate CNI with precision guarantee. At an abstract level, NeuCEPT-discovery considers each pair of a layer's activation output of the examined DNN as an input-response pair and conducts critical neuron selection on them. Different from the sequential formulation of CNI (4-2), NeuCEPT-discovery is executed in a pair-wise manner and, consequently, can be implemented efficiently. This section also provides theoretical results showing that the modification to pair-wise optimization still guarantees a specified precision level.

We now formalize the usage of the Markov blanket. Given a random variable T, we denote $\mathcal{M}_l(T) \subseteq \mathcal{N}_l$ as the smallest set of neurons at layer l such that, conditionally on the variables in that set - $Z_l^{\mathcal{M}_l(T)}$, T is independent of all other variables at layer l. In the studies of graphical models, the set $\mathcal{M}_l(T)$ is commonly addressed as the Markov blanket (MB) of T. We just make a slight modification by restricting the set of variables to a certain layer of the model. Under very mild conditions about the joint distribution of T and Z_l, the MB is well defined and unique [104]. We follow researchers in the field, assume these conditions [36], and proceed from there.

We have $\mathcal{S}_l = \mathcal{M}_l\left(Z_{l+1}^{\mathcal{S}_{l+1}}\right)$, the MB at layer l of $Z_{l+1}^{\mathcal{S}_{l+1}}$, achieves the maximum of the objective (4-2) since it contains all information about $Z_{l+1}^{\mathcal{S}_{l+1}}$. Thus, we have a straight approach to solve (4-2): Given the activation of interest at the last layer $Y = Z_L^{\{o\}}$, we solve for $\mathcal{M}_{L-1}(Y)$ - the MB at layer $L-1$. Then, at layer $L-2$, we find the MB of the variables in $\mathcal{M}_{L-1}(Y)$. The process continues until the first layer is reached. The computation can be described as:

$$\mathcal{S}_{L-1} \leftarrow \mathcal{M}_{L-1}(Y), \quad \mathcal{S}_{l-1} \leftarrow \mathcal{M}_{l-1}\left(Z_l^{\mathcal{S}_l}\right). \tag{4-3}$$

```
Input: Samples of model's activation $Z = (Z_1, ..., Z_M)$ at $M$ given layers. Precision thresholds $p = (p_1, ..., p_M)$.
Output: Estimation of critical neurons at all examined layers $\hat{\mathcal{M}}_1, \cdots, \hat{\mathcal{M}}_M$.
1: $Y \leftarrow Z_{M+1}^{\{o\}}$.
2: for $l = 1$ to $M$ do
3:     $\hat{\mathcal{M}}_l \leftarrow$ estimation of the Markov blanket of $Y$ at layer $l$ with precision control $p_l$.
4: end for
5: Return $\hat{\mathcal{M}}_1, \cdots, \hat{\mathcal{M}}_M$.
```

Algorithm 4-1. NeuCEPT-discovery.

Directly solving (4-3) is impractical as the problem is in NP-hard [88]. Additionally, estimating the distribution of $Z_l^{\mathcal{S}_l}$ via sampling is also impractical due to the curse-of-dimensionality. Our key observation to overcome those challenges is that the MB of the model's output variable Y at each layer l is a subset of $\mathcal{S}_l$ (Eq. (4-3)). As a result, given a solver solving for $\mathcal{M}_l\left(Y\right)$ with precision at least p, the output of that solver is also an approximation of $\mathcal{S}_l$ with the precision at least p. This allows us to solve for $\mathcal{M}_l\left(Y\right)$ instead of $\mathcal{S}_l$ and overcome the high-dimensionality of $Z_l^{\mathcal{S}_l}$. This observation is exploited in the NeuCEPT-discovery step of the proposed algorithm, which is described in Algorithm 4-1. The proof that NeuCEPT-discovery achieves precision guarantee is based on the following Theorem 4-1.

Theorem 4-1. *Suppose we have a solver solving for the MB of a set of random variables and apply that solver to each layer of a neural network as described in equation (4-3), then the solution returned by the solver at each layer must contain the MB of the neural network's output at that layer, i.e.* $\mathcal{M}_l(Y) \subseteq \mathcal{S}_l$, $\forall l = 0, ..., L-1$.

Proof. For simplicity, we consider the following Markov chain $Z_0 \rightarrow Z_1 \rightarrow Y$. We now show that $\mathcal{M}_0(Y) \subseteq \mathcal{M}_0\left(Z_1^{\mathcal{M}_1(Y)}\right)$. We have:

1. Z_1 determines Y and $Y \perp\!\!\!\perp \left\{Z_1 \setminus Z_1^{\mathcal{M}_1(Y)}\right\} | Z_1^{\mathcal{M}_1(Y)}$ so that $Z_1^{\mathcal{M}_1(Y)}$ determines Y. We can also write this statement as $Z_1^{\mathcal{S}_1}$ determines Y.

2. Z_0 determines $Z_1^{\mathcal{M}_1(Y)}$ and

$$Z_1^{\mathcal{M}_1(Y)} \perp\!\!\!\perp \left\{ Z_0 \setminus Z_0^{\mathcal{M}_0\left(Z_1^{\mathcal{M}_1(Y)}\right)} \right\} | Z_0^{\mathcal{M}_0\left(Z_1^{\mathcal{M}_1(Y)}\right)}.$$

so that $Z_0^{\mathcal{M}_0\left(Z_1^{\mathcal{M}_1(Y)}\right)}$ determines $Z_1^{\mathcal{M}_1(Y)}$. We can write this statement as $Z_0^{\mathcal{S}_0}$ determines $Z_1^{\mathcal{S}_1}$.

3. Combine the two above statements, we have $Z_0^{\mathcal{M}_0\left(Z_1^{\mathcal{M}_1(Y)}\right)}$ determines Y.

On the other hand, we have $\mathcal{M}_1(Y)$ is the smallest subset of neurons at the Z_0 layer that determines Y. Due to the uniqueness of the minimal set that separates Y from the rest of the variables (which is the MB of Y) [104], we have $\mathcal{M}_1(Y) \subseteq \mathcal{M}_1\left(Z_2^{\mathcal{M}_2(Y)}\right)$.

The proof generalizes for the case of L layers Markov chain $Z_0 \to Z_1 \to \cdots \to Z_L$ as the same arguments can be used to show that $Z_l^{\mathcal{S}_l}$ determines $Z_{l+1}^{\mathcal{S}_{l+1}}$. This would lead to the fact that all $Z_l^{\mathcal{S}_l}$ can determine Y; hence, $\mathcal{S}_l$ contains $\mathcal{M}_l(Y)$ due to the uniqueness of the MB [104]. □

We now can formalize and prove the statement that any MB solvers with a precision guarantee p on the input-response pair (Z_l, Y) can be used to solve for the MB of the pair $(Z_l, Z_{l+1}^{\mathcal{S}_{l+1}})$ with precision at least p in the following Corollary 1:

Corollary 1. *Suppose we have a solver solving for the MB of a random response T with the precision at least p for a given $0 < p < 1$. Let $\hat{\mathcal{M}}_l$ be the output of that solver on the input-response pair (Z_l, Y) defined in procedure (4-3). Then, $\hat{\mathcal{M}}_l$ also satisfies the precision guarantee p as if we solve for the input-response pair $(Z_l, Z_{l+1}^{\mathcal{S}_{l+1}})$.*

Proof. Denote $q = 1 - p$. Since the precision is one minus the FDR, we can instead prove:

$$\text{FDR} := \mathbb{E}\left[\frac{\#\{j : j \in \hat{\mathcal{M}}_l \setminus \mathcal{S}_l\}}{\#\{j : j \in \hat{\mathcal{M}}_l\}}\right] \leq q. \tag{4-4}$$

From Theorem 4-1, we have $\mathcal{M}_l(Y) \subseteq \mathcal{S}_l$ for all $l = 0, \cdots, L-1$. This implies:

$$\begin{aligned} &\hat{\mathcal{M}}_l \setminus \mathcal{S}_l \subseteq \hat{\mathcal{M}}_l \setminus \mathcal{M}_l(Y) \\ \Longrightarrow &\#\{j : j \in \hat{\mathcal{M}}_l \setminus \mathcal{S}_l\} \leq \#\{j : j \in \hat{\mathcal{M}}_l \setminus \mathcal{M}_l(Y)\} \end{aligned} \tag{4-5}$$

On the other hand, as $\hat{\mathcal{M}}_l$ is the solution of the solver on the input-response pair (Z_l, Y) with FDR less than or equal to q:

$$\mathbb{E}\left[\frac{\#\{j : j \in \hat{\mathcal{M}}_l \setminus \mathcal{M}_l(Y)\}}{\#\{j : j \in \hat{\mathcal{M}}_l\}}\right] \leq q. \tag{4-6}$$

Combining (4-5) and (4-6), we have the Corollary. □

Corollary 1 enables NeuCEPT to exploit any solver with precision control to efficiently solve for the procedure (4-3) with precision guarantee. The current implementation of NeuCEPT-discovery uses Model-X Knockoffs [17]. The following paragraphs provide the description of the solver.

Model-X Knockoffs is a new statistical tool investigating the relationship between a large set of explanatory variables and a response T. It considers a very general conditional model, where T can depend in an arbitrary fashion on the variables' covariates $R = (R_1, \cdots, R_p)$. From a set of hundreds or thousands of variables, Model-X Knockoffs can identify a smaller subset potentially explaining the response while rigorously controlling the False-discovery-rate (FDR), which will be defined in more detail (Eq. (4-4)).

Specifically, for each sample of R, the Model-X Knockoffs generates a knockoff copy $\tilde{R}$ satisfying $Y \perp\!\!\!\perp \tilde{R}|R$ and the *pairwise exchangeable* property [17]. Then, the importance measure U_j and $\tilde{U}_j$ are computed for each R_j and $\tilde{R}_j$, respectively. After that, the statistics $W_j = U_j - \tilde{U}_j$ is evaluated for each feature. A large positive value of W_j implies evidence against the hypothesis that the j^{th} feature is not in the Markov blanket of T. The work [17] has shown that exactly controlling the FDR below the nominal level q can be obtained by selecting $\hat{\mathcal{R}} = \{j : W_j \geq \tau_q\}$, where τ_q is defined as:

$$\tau_q = \min\left\{t > 0 : \frac{1 + |\{j : W_j \leq t\}|}{|\{j : W_j \geq t\}|} \leq q\right\}.$$

In this paper, we use $(.,.)$ to denote the input-response pair for Model-X Knockoffs as a general solver for the Markov blanket, e.g. (R, T) for the formulation above.

4.2 Information-Theoretic Interpretation and NeuCEPT-Learning of Critical Neurons

The goal of finding critical neurons is to correctly identify the model's mechanisms. Sect. 4.2.1 discusses in more detail the mechanisms and how the MI objective in Eq. (4-2) is apt for the task. Sect. 4.2.2 describes how NeuCEPT extracts information from critical neurons to identify the model's mechanism.

4.2.1 Information-Theoretic Interpretation

The information-theoretic interpretation of critical neurons is formulated via the *mechanism* of the predictions, the *explainability power* of the neurons, and the *non-redundancy* property.

Mechanism: Previous analysis [12] reveals distinctive patterns of neurons' activation shared among some input samples. This similarity suggests they might be processed in the same manner by the model, which is what we call *mechanism*. Similar to how unlabeled data is handled in unsupervised learning, the mechanism in this work is modeled as a discrete latent random variable whose realization determines how the predictions are generated. Fig. 4-4 provides an intuition on the relationship between the neurons' activation and mechanisms under this assumption. Suppose the latent mechanism variable C determines the generation of predictions of the class *goldfish*. Different realizations of C, i.e. 0 or 1, result in different patterns in the activation Z. On one hand, these patterns specify how the model predicts. On the other hand, observing the activation of some neurons, i.e. critical neurons, can be sufficient to determine the realization of C, i.e. the model's underlying mechanism.

Explainability power: An intuitive necessary condition on the selection of critical neurons is that their activation should determine (or significantly reduce the uncertainty of) the mechanism C. This condition is called *explainability power*. To see how the objective (4-2) fits into this condition, let's consider its MB solution $\{\mathcal{S}_l\}_{l=0}^{L-1}$. From the definition of the MB, for any set of neurons $\mathcal{R}_l$ at a layer l that is disjoint with $\mathcal{S}_l$, we have $Z_l^{\mathcal{R}_l}$ is independent with $Z_{l+1}^{\mathcal{S}_{l+1}}$ given $Z_l^{\mathcal{S}_l}$. Since $Z_{l+1}^{\mathcal{S}_{l+1}}$ determines $Z_{l+k}^{\mathcal{S}_{l+k}}$ for all $k > 1$, variables in $\mathcal{R}_l$ must also independent with $Z_{l+k}^{\mathcal{S}_{l+k}}$ given $Z_l^{\mathcal{S}_l}$. Thus, knowing $Z_l^{\mathcal{R}_l}$ does not provide any additional information on how the model generates the prediction of interest, i.e. $\{\mathcal{S}_l\}_{l=0}^{L-1}$ is sufficient.

Non-redundancy: Neurons' activation not only determines the mechanisms and the predictions but also contains other information on the data. However, not all information stored in the activation is necessarily used by the model in generating the examined predictions. Thus, another desirable property of the identified neurons is that they must be used by the model in generating the predictions of interest. We call this condition *non-redundancy*. Intuitively, selecting more neurons gives us more explainability power and less non-redundancy.

To demonstrate how the proposed objective (4-2) meets the notion of non-redundancy, we compare it with another objective aiming to identify the set of globally important neurons derived from the Markov chain (4-1):

$$\mathcal{S}_l^* = \text{argmax}_{\mathcal{S} \subseteq \mathcal{N}_l} I\left(Z_l^{\mathcal{S}}; Z_{l+1}, \ldots, Z_L\right), \text{ s.t. } \mathcal{S} \in \mathcal{C}. \tag{4-7}$$

This objective tells us how much we know about the model's later layers given the activation of the neurons in $\mathcal{S} \subseteq \mathcal{N}_l$. From the lens of information theory, information encoded in $\{\mathcal{S}_l^*\}_{l=0}^{L-1}$ can fully determine all information of the models and satisfies the notion of explainability power. However, for a specific prediction, that information is too excessive since the objective considers all classes equally. Mechanisms discovered on top of them are prone to redundancy. This reason motivates us to use the objective (4-2). This also demonstrates the importance of the critical neuron discovery step in NeuCEPT.

4.2.2 NeuCEPT-Learning on Critical Neurons

Given the critical neurons identified by NeuCEPT-discovery, NeuCEPT-learning extracts their activation to identify the model's mechanisms. Since the ground-truth mechanisms are not given, it is natural to consider the mechanism identification/discovery problem as an unsupervised learning task, which has been extensively studied [49].

Algorithm 4-2 describes how NeuCEPT extracts information from critical neurons. Besides their activation, the algorithm's inputs include a set of compactness parameters limiting the number of representative neurons and an integer K guessing the number of mechanisms. The usage of the compactness parameters is common in many existing explanation methods for the

Input: Critical neurons' activation at M examined layers, denoted as $Z^{\mathcal{S}}$, the number representative neurons $\{v_i\}_{i=1}^{M}$ and the number of clusters/mechanisms K.
Output: The explained representations of all inputs and their corresponding mechanisms/clusters.

```
1: # *Constraints enforcement*
2: **for** $l = 1$ to $M$ **do**
3:     $V_l \leftarrow$ Feature Agglomeration ($Z_l^{\mathcal{S}_l}$) with constraint $|V_l| \leq v_l$ or selecting top $v_l$ neurons.
4: **end for**
5: # *Unsupervised learning*
6: $g \leftarrow$ Initialize an unsupervised clustering model with $K$ components.
7: Fit $g$ on $V = (V_0, \cdots, V_{L-1})$.
8: Mechanism $C \leftarrow g(V)$.
9: Return $C$.
```

Algorithm 4-2. NeuCEPT-learning.

sake of visualization. The first step of NeuCEPT-learning is to enforce this compactness requirement, i.e. either by selecting the top neurons identified in NeuCEPT-discovery or by feature-agglomerating those neurons into a smaller set of representative neurons. Similar techniques have been used to apply Model-X Knockoffs on real-data with very high-correlated features [17]. Then, an unsupervised learning method is chosen among K-means, Gaussian Mixture, and Agglomerative Clustering to map each input sample to one of the K clusters representing K mechanisms.

This section is concluded with a demonstration of some NeuCEPT's outputs in analyzing predictions of Inception-v3. The examined layers are the last layers of the *Mixed-5d* and the *Mixed-6e* blocks (Fig. 4-5). The numbers of representative neurons are restricted to 5 and 3. Next to each input, we show a graph representing the activation's level (red for high, blue for low) of those representative neurons from the *Mixed-5d* (left) to the *Mixed-6e* (middle). The last dot represents the output neuron (right). NeuCEPT-learning helps us visualize similar activation's patterns among samples of the same mechanism, and differentiate them from another mechanism.

4.3 Experiments: Setting and Results

The experiments in this section focus on evaluating the explainability power and the non-redundancy properties, mentioned in Sect. 4.2.1. While evaluating the explainability power can be conducted via ablation study, evaluating the non-redundancy is more challenging as we normally do not know the underlying mechanism. To tackle this, we propose a training setup, called *prior-knowledge training*, so that both the non-redundancy and the explainability power can be evaluated.

4.3.1 Prior-Knowledge Training (PKT)

The PKT is introduced so that models with partially known mechanisms can be obtained. Specifically, during the PKT, certain prior-knowledge is injected into a model, called PKT model, before the training of its main task, called *posterior-task*. Due to the injection, those mechanisms are expected to be embedded and used by the PKT model to conduct its posterior-task. With that knowledge on the PKT model, we then can evaluate explaining methods accordingly.

Fig. 4-7 describes the PKT in more detail. First, a model called *prior-model*, is trained on a more specific task, called *prior-task*. The prior-task's training data is chosen such that there exists information that is only available to the prior-model (therefore the term *more informative dataset*). To ensure that information is embedded in the prior-model, the prior-task is the task of predicting that information. The weights of the prior-model is then transferred, completely or partially, to the PKT model before its training on the posterior-task. As such, given a PKT model and a conventionally trained model, a good mechanism discovery algorithm should claim that the PKT model relies on some specific information, i.e. the prior-knowledge, to generate its prediction (explainability power) while the conventionally trained one does not (non-redundancy). In fact, given a cluster label c returned by the algorithm and the label y_{prior} of the informative dataset, the algorithm can be evaluated by the clusters' entropy (CE) metric:

$$\textstyle\sum_c \sum_{y_{prior}} p(y_{prior}, c) \log\left(1/p(y_{prior}|c)\right),$$

where $p(.)$ is the empirical probability. The lower the CE, the more the clusters/mechanisms identified by the algorithm align with the prior-knowledge. Thus, the CE of the PKT model resulting from a good mechanism discovery algorithm should be lower than that of a conventionally trained model.

Readers can refer to Fig. 4-2 for a better understanding of PKT. In that experiment, the prior-task and the posterior-task are the digit classification and the *even/odd* classification, respectively. The models, from left-to-right, are the prior-model, the PKT model and the conventionally trained model.

4.3.2 Experimental Setting

We now describe the experimental settings of this work. Regarding the hardware, The experiments are implemented in Python 3.8 and conducted on a single GPU-assisted compute node with a Linux 64-bit operating system. The allocated resources include 32 CPU cores (AMD EPYC 7742 model) with 2 threads per core, 8 GPUs (NVIDIA DGX A100 SuperPod model) with 80GB of memory per GPU, and 100GB of RAM.

Regarding the dataset, the experiments are conducted on LeNet [78] and VGG [84] trained on MNIST [79] and CIFAR10 [74] dataset, respectively. The models' configurations follow the default setting of Pytorch [101] with slight modification at the last layer to fit the tasks. Specifically, the output layers of the normal and the PKT model are changed to 2 neurons so that they predict the *even/odd* in MNIST and *animal/object* in CIFAR10. Note that, in training the PKT model on CIFAR10, we freeze the parameters of the first 6th convolutional layers (there are 9 layers) to better maintain the prior-knowledge/mechanisms transferred from the prior model.

To our knowledge, there exists no work directly addressing the proposed problem to serve as the baseline. As such, we adopt state-of-the-art local explanation methods, including Saliency [123], Integrated-Gradients (IG) [131], Deeplift [122] and Gradient-SHAP (G-SHAP) [86], implemented by Captum [1], to identify critical neurons. Specifically, the neurons are selected by their total attribution scores given by the explanation methods on all images of the examined class.

For the choice of precision values in our method, there are three deciding factors: the layer's location, the number of neurons in that layer, and the model's complexity. The general intuition is, that the deeper the layer, the smaller of number of neurons in that layer, and the less complex the model, the easier the critical neurons can be identified. Thus, in such cases, we can set a high value of precision and expect good solutions. In experiments for LeNet, the precisions are between 0.9 and 0.98. For CIFAR10, the values are between 0.4 and 0.8 based on the layer and the number of neurons. In Inception-v3 and CheXNet, the values are 0.6 for all layers.

Regarding the usage of the Model-X Knockoffs, there is a major issue when applying the method to high-dimensional real data, specifically, on neurons' activations. The challenge arises when we need to choose between two or more very highly correlated features. Note that "this is purely a problem of power and would not affect the Type I error control of knockoffs" [17]. To overcome this, the method clustered features using estimated correlations as a similarity measure. After that, one representative feature is chosen from each cluster. The representatives are then used in the feature-discovery task instead of the original features. The feature agglomeration step (described in Sect. 4.2 and Algo. 4-2) is based on this alleviation.

4.3.3 Explainability Power

High explainability power implies the selected neurons hold more information determining the mechanism. Thus, the CE should be small if the guessing number of clusters K (specified in Sect. 4.2.2) aligns with the actual number of the mechanisms. In the posterior-tasks of MNIST and CIFAR10, we train LeNet and VGG to classify *even/odd* and *animal/object*, respectively. The actual numbers of the mechanisms in the PKT models are expected to be at least $(5,5)$ and $(4,6)$ for each pair of labels in the task. The reason is they are the number of original labels belonging to those categories, i.e. 5 odd digits, 5 even digits, 4 animals, and 6 objects.

Fig. 4-7 shows the CE of the clusters learned on neurons identified by different explanation methods versus the number of clusters K. MNIST's results clearly show neurons identified by NeuCEPT can differentiate the inputs based on their original labels embedded by prior-knowledge training. Notably, when $K = 5$, NeuCEPT achieves its lowest, which aligns with our expectation on the number of actual mechanisms that the PKT model uses. VGG's experiments show similar results with less distinction in the number of clusters.

Since the mechanism specifies the prediction, critical neurons should have a high impact on model's predictions. This impact can be evaluated via ablation test [99], in which noise of different levels and configurations are added to those neurons and the model's accuracy is recorded accordingly. If the neurons have explainability power, protecting them from the noise should maintain the model's accuracy [99, 142].

Fig. 4-8 and 4-9 show the ablation tests of neurons identified by different methods. In the figures, *Top* is the number of neurons protected from the noise, determined by the score of the explanation methods. γ is an exponential decay parameter determining how noise is distributed among neurons: the larger the γ, the lesser the noise added to the protected neurons. The tests are conducted at different layers of LeNet and VGG. The results show that neurons identified by NeuCEPT hold higher predictive power among most of the experiments

Fig. 4-11 shows the adaptation of NeuCEPT to LeNet to select important input features for each class. The purpose of this experiment is to demonstrate that NeuCEPT can adapt to the concerns regarding Model-X Knockoffs on high-dimensional highly-correlated data and unknown covariance [17]. The plots show that the explainability power of features selected by NeuCEPT is comparable with other methods.

4.3.4 Non-Redundancy.

The following experiments show that NeuCEPT meets the non-redundancy requirement, i.e. it should return a significantly higher CE when examining conventionally trained models, which have no knowledge of the original labels. Fig. 4-10 plots the differences in the CE between the conventionally trained models and the PKT models on MNIST and CIFAR10. While the differences in CE at certain layers of some other methods fluctuate around 0, indicating there is no difference between the two models, NeuCEPT consistently supports the expectation that only the PKT models recognize the prior-knowledge and the conventionally trained models do not. Note that NeuCEPT differentiates the models simply by observing their activation on critical neurons. This means those critical neurons are indeed used by the model in generating the examined predictions.

4.3.5 Running-Time.

Table 4-1 shows the average running time of all methods in searching for the important neurons for a single class. All experiments are on 10000 test samples. A straight comparison among methods is not trivial due to the difference in resource utilization and hyper-parameter selections. Technically, NeuCEPT only needs to run Model-X Knockoffs once; however, we run it 50 times for more stable results. The main takeaway is that NeuCEPT can be run in a reasonable amount of time on moderate-size models such as VGG.

4.4 Case Studies

This section provides some usages of NeuCEPT in analyzing different interesting aspects of DNNs: the linear separability (studied along with the Linear Probe), the explainability (studying on Inception-v3 model), and the predictions' reliability (studying on CheXNet model).

Linear Probe (LP) [2] are linear classifiers hooking on intermediate layers of DNNs to measure their linear separability. Intuitively, a low loss of the probe in predicting some labels at a layer implies that the activation of that layer is more linearly separable. This metric is important in the analysis of DNNs since it can be used to characterize layers, debug models, or monitor training's progress.

We applied LP to LeNet on the MNIST dataset (Fig. 4-2) and plotted the results in Fig. 4-12. The notations (2), (10), and NeuCEPT refer to the labels that the LPs predict: (2) is the model's labels *even/odd*, (10) is the original digit labels and NeuCEPT is the mechanisms identified by NeuCEPT. From the bottom visualization of Fig. 4-2, LeNet(A) is expected to be more linearly separable as its point-cloud is divided into more distinctive clusters. However, simply applying LP on the model's labels (notation (2)) shows little difference between the two models. On the other hand, both results of the LPs with the digit labels (notation (10)) and with the unsupervised clusters/mechanisms learnt by NeuCEPT state that LeNet(A) is more linearly separable. Note that the gap between LeNetA(10) and LeNetB(10) cannot be obtained by LP in practice due to the lack of ground-truth knowledge. This example shows how NeuCEPT can be used to strengthen the results obtained by LP and further the study of linear separability.

As attribution methods mainly attribute scores to neurons with the most contribution to the prediction, aggregating the resulted scores among inputs of a class naturally gives highly attributing neurons of the class. However, as the activation of those neurons can be highly similar among samples of the class (a trivial example is the output's neuron associated with the class), they hold limited information on how the model processes its inputs differently and, hence, do not fit for the task of identifying mechanisms.

The analysis of the class *goldfish* and *bee* of Inception-v3 [132] using NeuCEPT (Fig. 4-13) demonstrates the above claim. Specifically, at the *Mixed6e* block of the model, NeuCEPT identifies a neuron that is not among the top highly activated of the examined class but holds valuable information about how the model processes inputs of that class. In fact, for each class, there are two subsets of inputs such that members of one subset activate the neuron while those of the other do not. Simply using activation's level, which can be considered as the simplest form of attribution method, would miss this neuron. Interestingly, there are distinctive visual concepts associated with images belonging to clusters identified by NeuCEPT. In the class *goldfish*, one subset is about *a single fish* while the other is about *a shoal of fishes*. For the class *bee*, it is *a single bee* versus *a single flower* or *a bunch of flowers*. This observation supports the hypothesis that the two subsets of images are indeed processed differently by the model.

CheXNet is a modern DNN that can detect pneumonia from chest X-rays at a level exceeding practicing radiologists [108]. However, the work [158] found out that the model has learned to detect a hospital-specific metal token on the scan to generate some classifications. This finding raises an important question: are there systematical methods to differentiate reliable predictions from unreliable ones?

We partially address that question using NeuCEPT (Fig. 4-14): we discover a subclass of 150 unreliable *'pneumonia'* predictions with much lower precision than that on the whole class (2978 samples), i.e. 36.7% compared to 60.6%. We then use the local explanations provided by the model itself to further verify our findings. It can be observed that false-positive predictions generally are made using features outside of the lung's area. This example demonstrates that identifying mechanisms underlying the model's predictions can help evaluate the reliability of individual predictions.

4.5 Conclusion

This work aims to learn the mechanisms underlying DNNs' predictions to provide a deeper explanation of how the models work. From an information-theoretic viewpoint, the problem is formulated as a sequence of MI maximization, whose solution, called critical neurons, can be solved by our NeuCEPT-discovery with guarantee. We develop NeuCEPT-learning, an algorithm clustering inputs based on their activation of critical neurons, to reveal the model's mechanisms. We further designed a training procedure so that the mechanism discovery task can be evaluated. The experiments and case studies show that NeuCEPT consistently identifies the underlying mechanisms and reveals interesting behaviors of the DNNs.

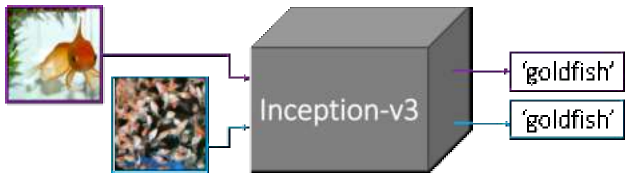
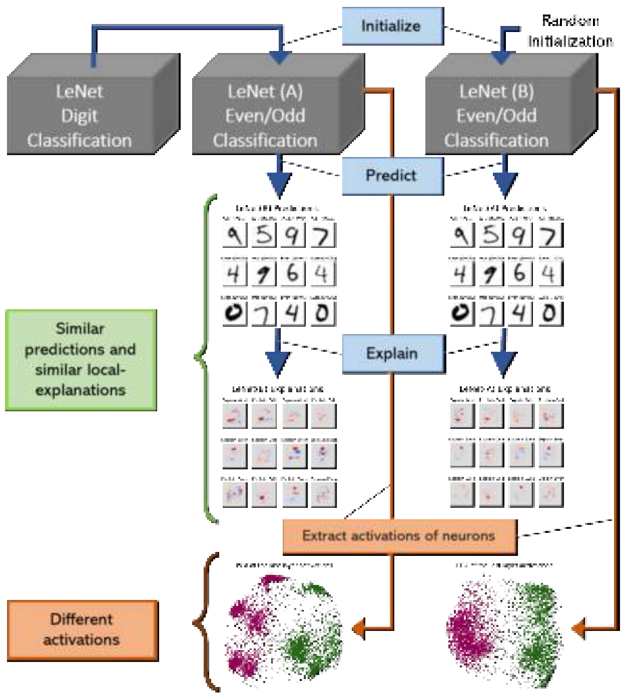


Figure 4-1. Two images have the same predictions *goldfish* generated by Inception-v3. One is a *single goldfish* while the other is a *shoal of fish*. Are the mechanisms behind the two predictions the same?



Initialize
Random Initialization
LeNet Digit Classification
LeNet (A) Even/Odd Classification
LeNet (B) Even/Odd Classification
Predict
Explain
Similar predictions and similar local-explanations
Extract activations of neurons
Different activations

Figure 4-2. LeNet(A) is initialized by a LeNet pretrained on digit classification task and LeNet(B) is initialized randomly. While the *even/odd* predictions and the explanations provide little information differentiating the two models, extracting and visualizing the activation of some neurons at their last layers reveal that LeNet(A) groups input into more distinctive clusters with the same ground-truth digit-labels of the dataset. More details of the experiments are in Sect. 4.3.1

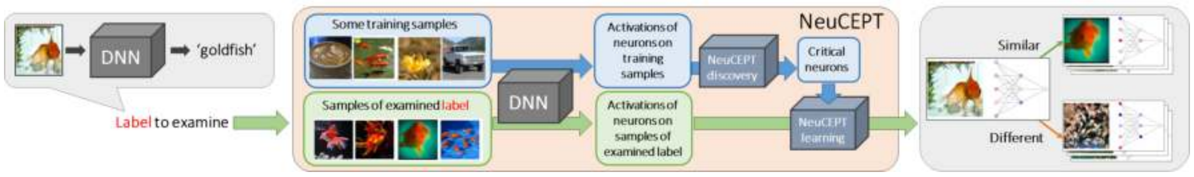


Figure 4-3. Overall architecture of NeuCEPT. Given a prediction of a class of interest, NeuCEPT collects the model's inputs from that class. Then, by forwarding them through the DNN, NeuCEPT obtains the activation and solves the set of critical neurons at some layers. Finally, mechanisms are learnt via unsupervised learning on those neurons' activation.

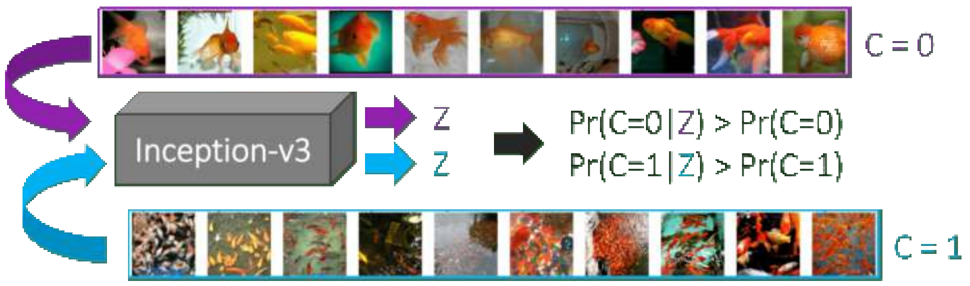


Figure 4-4. The probabilistic intuition of the neurons' activation and mechanism: the mechanism determines the activation, and, conversely, observing the activation can reveal the mechanism.

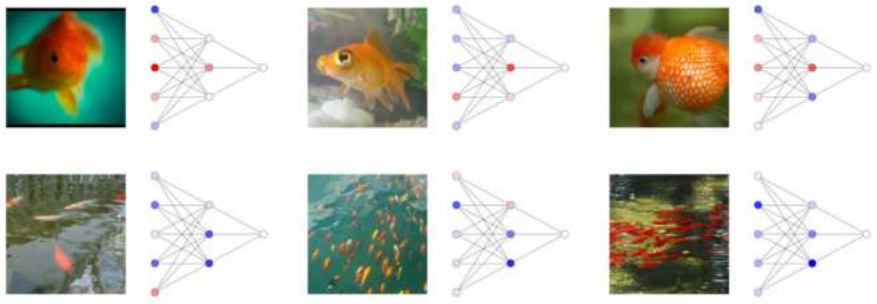


Figure 4-5. Examples of NeuCEPT's outputs on Inception-v3. Images from each row are from the same cluster learnt by NeuCEPT.

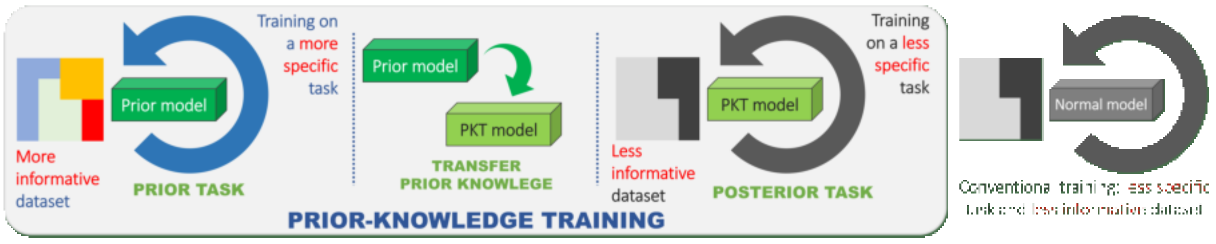


Figure 4-6. Prior-knowledge training: parameters of the prior-model trained on a more informative task are transferred to the PKT model, which is later trained on a less specific task. The mechanisms of the PKT model is expected to be different from normally-trained models.

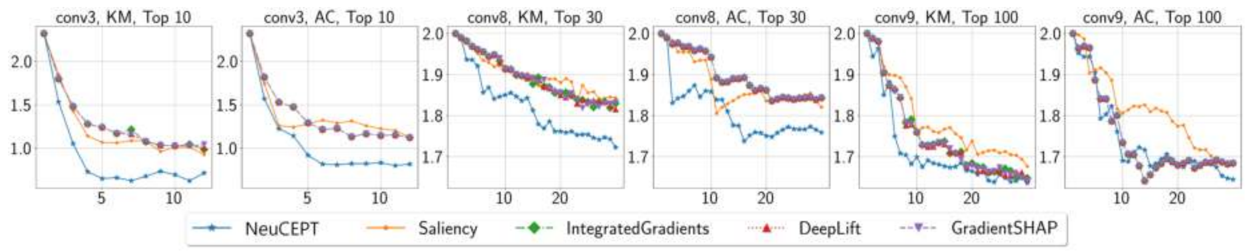


Figure 4-7. CE (bits) resulted from K-mean (KM) and Agglomerative clustering (AC) clustering on activation of neurons selected by different methods. The x-axis and y-axis are the number of clusters and the CE, respectively. The first 2 plots are the analysis at layer *conv3* of the LeNet on the class *even* using top-10 neurons. The last 4 plots are of VGG on class *object* with top 30/100 neurons at layer *conv8* and *conv9*.

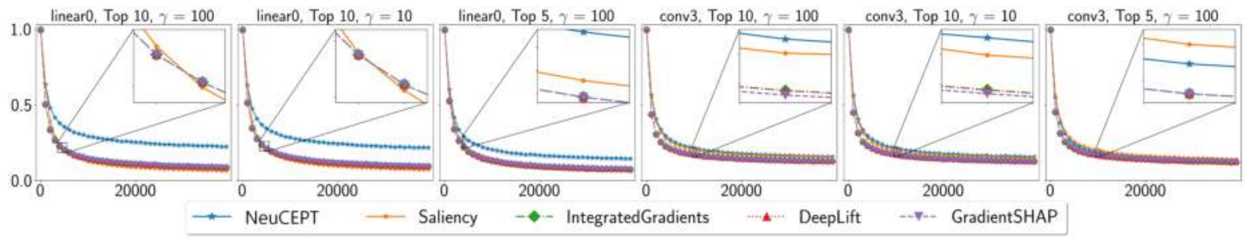


Figure 4-8. Ablation tests on LeNet. The x-axis and y-axis are the noise levels and the test accuracy.

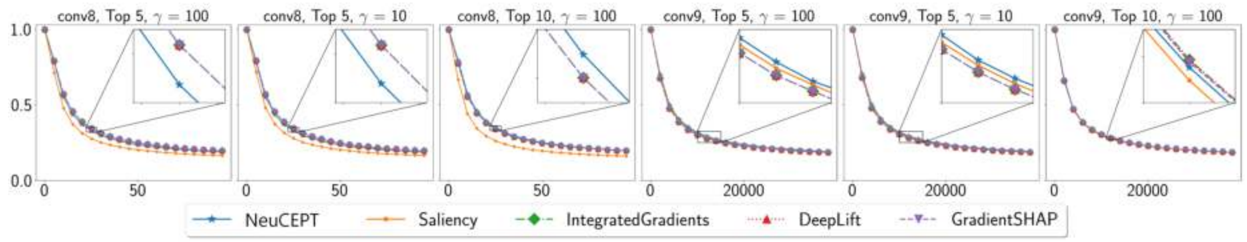


Figure 4-9. Ablation tests on VGG. The x-axis and y-axis are the noise levels and the test accuracy, respectively.

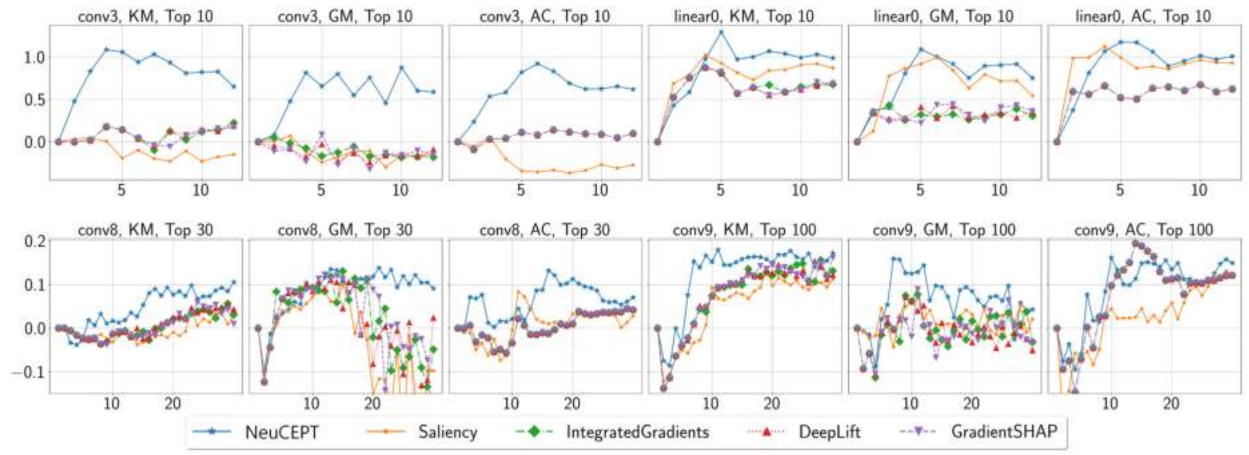


Figure 4-10. The CE differences (bits) between the conventionally trained model and the PKT model. Clusters are learned using k-mean (KM), Gaussian Mixture (GM), and Agglomerative clustering (AC) on the activation of neurons chosen by different methods. The x-axis and y-axis are the number of clusters and the CE differences. The top figures show the results of the LeNet at different layers with the top 10 important neurons. Bottom are the results of VGG with top 30/100 neurons at convolutional layers 8 and 9.

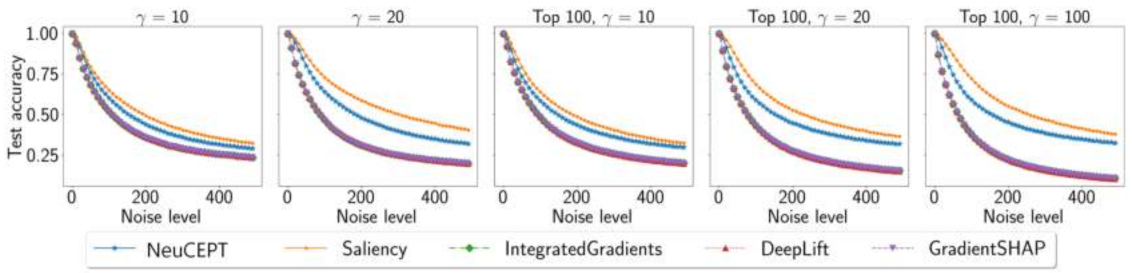


Figure 4-11. Ablation results at the input layer of LeNet. The two plots on the left show the ablation results with continuous noise, i.e. the noise is added based on the score given by the associated methods. The three figures on the right show results when the top 100 features are selected. γ is an exponential decay parameter determining how noise is distributed among neurons.

Table 4-1. The running time in seconds of tested methods. For the 4 other explanation methods, we use the Captum library which runs on GPUs. Our NeuCEPT runs entirely on CPU cores and the reported running-time is for one iteration. For more stable results, the experimental results of NeuCEPT are obtained after 50 iterations.

Model	Layer	NeuCEPT	Saliency	IG	DeepLift	G-SHAP
LeNet prior	conv3	3.18	17.35	79.59	51.32	31.42
	linear0	1.93	16.25	79.20	20.12	31.13
LeNet normal	conv3	2.86	16.52	79.95	51.22	31.03
	linear0	2.00	15.47	77.57	19.93	31.08
LeNet PKT	conv3	2.76	16.00	80.46	50.49	31.93
	linear0	1.94	15.53	77.13	20.05	30.77
VGG11 prior	conv8	20.86	167.82	379.16	195.01	355.65
	conv9	22.36	180.6	362.28	183.97	305.65
VGG11 normal	conv8	26.12	307.74	709.38	335.29	708.50
	conv9	23.22	307.87	708.71	309.38	707.38
VGG11 PKT	conv8	17.26	223.24	457.78	263.63	457.09
	conv9	18.80	222.50	457.81	220.53	457.32

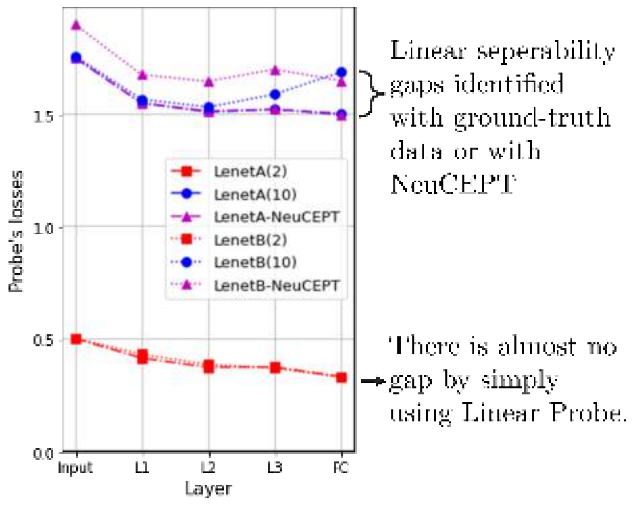
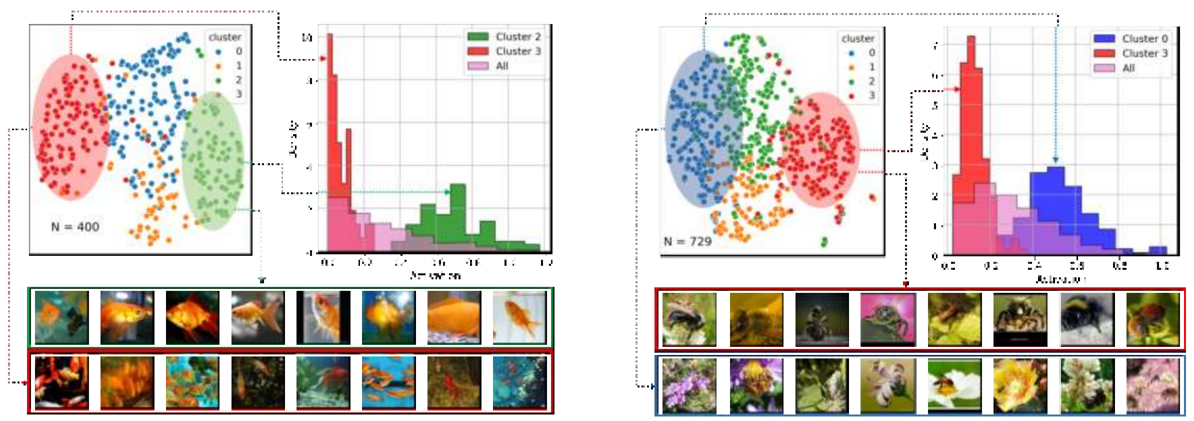


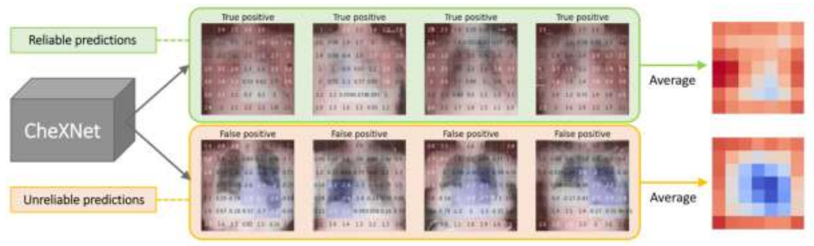
Figure 4-12. Using LP on the predicted labels, i.e. *even/odd* (indicated by the notation (2)), does not reveal the difference in the linear separability of LeNetA and LeNetB. However, with the aiding of the ground-truth labels or NeuCEPT, LP can detect the linear separability of the two models (indicated by the notation (10) and notation NeuCEPT).



Neuron 322 differentiates predictions of two sub-classes (red/green) of the class *goldfish*.

Neuron 305 differentiates predictions of two sub-classes (blue/red) of the class *bee*.

Figure 4-13. Results obtained by applying NeuCEPT with $K = 4$ on classes *goldfish* and *bee* at the *Mixed6e* block of Inception-v3. For each class, we use NeuCEPT to identify a neuron whose activation's histogram of 2 distinctive clusters (among 4) is shown on the top-right of the sub-figures. While these neurons are not among the top-activated neurons of each class, they hold vital information in differentiating the predictions as inputs of one cluster activate the neuron and the others do not. The top-left and the bottom figures plot the low-dimensional representations of the inputs and some images drawn from each cluster for visualization.



Experiment's info

Model	CheXNet
Examined predictions	*'Pneumonia'*
Total samples	2978
Positive samples	1035
Positive predictions	1035
Model's precision	60.6%
Reliable-subclass precision	91.3% (180 samples)
Unreliable-subclass precision	36.7% (150 samples)

Figure 4-14. NeuCEPT discovers a subclass of unreliable predictions with precision of 36.7% (compared to 60.6% on the whole class) in CheXNet. Explanation heat-maps provided by the model also suggest that unreliable predictions are made by the area at the boundary, not the lung's area of the image (red means high importance score).

CHAPTER 5
PGM-EXPLAINER: PROBABILISTIC GRAPHICAL MODEL EXPLANATIONS FOR GRAPH NEURAL NETWORKS

Graph Neural Networks (GNNs) have been emerging as powerful solutions to many real-world applications in various domains where the datasets are in form of graphs such as social networks, citation networks, knowledge graphs, and biological networks [156, 160, 163]. Many GNN's architectures with high predictive performance should be mentioned are ChebNets [27], Graph Convolutional Networks [71], GraphSage [55], Graph Attention Networks [137], among others [81, 94, 150, 154, 149, 80].

As the field grows, understanding why GNNs made such decisions becomes more vital. (a) It improves the model's transparency and consequently increases trust in the model. (b) Knowledge of the model's behaviors helps us identify scenarios in which the systems may fail. This is essential for safety reasons in complex real-world tasks in which not all possible scenarios are testable. (c) Due to fairness and privacy reasons, knowing if a model has bias in its decision is crucial. Although there are protections for specific classes of discrimination, there might be other unwanted biases [31]. Understanding the model's decisions helps us discover these biases before its deployment.

Although generating explanations for Conventional Neural Networks (CNNs) has been addressed by many methods and standardized tool-kits, called *explainers* [123, 111, 86, 121, 122, 9, 128, 64], the counterparts for GNNs are lacking. Until very recently, GNNExplainer [155], has been introduced to explain GNNs using a mutual-information approach. As this is a pioneer in explaining GNNs, there is little knowledge on the quality of GNNExplainer and it is unclear whether mutual-information is apt for the task. Furthermore, GNNExplainer requires an explained model to evaluate its prediction on the fractional adjacency matrix. However, most available libraries for GNNs, such as Pytorch [101] and DGL [146], do not meet this requirement as the matrix is used to compute the discrete sum in the messages passing steps. Another work in [106] adapts several existing gradient-based explanation methods for CNNs [123, 121, 64] to GNNs settings. Nevertheless, these methods not only require knowledge on the internal parameters of the model but also are not specifically designed for GNNs. Additionally, all of the above methods fall into a class of explanation methods, named *additive feature attribution methods* [86], which is based on the linearly independent assumption of explained features.[1] However, due to non-linear activation layers, GNN integrates input features in a non-linear manner. Relying on linearly independent assumptions to generate explanations for GNNs, where explained features can be highly dependent on each other, might degrade the explanation's quality significantly.

We propose a Probabilistic Graphical Model model-agnostic explainer for GNNs, called PGM-Explainer. In PGM-Explainer, the explanation of a GNN's prediction is a simpler interpretable Bayesian network approximating that prediction. Since Bayesian networks do not rely on the linear-independence assumption of explained features, PGM-Explainer is able to illustrate the dependency among explained features and provide deeper explanations for GNNs' predictions than those of additive feature attribution methods. We further provide theoretical analysis show that, if a perfect map for the sampled data generated by perturbing the GNN's input exists, a Bayesian network generated by PGM-Explainer always includes the Markov-blanket of the target prediction. This means the resulting PGM contains all statistical information on the explained prediction encoded in the perfect map. PGM-Explainer is evaluated on synthetic datasets and real-world datasets for both node and graph classification. The evaluations based on ground-truth explanations and human-subjective tests demonstrate PGM-Explainer provides accurate and intuitive explanations for the predictions.

Section 5.1 provides some preliminaries, including an explanation model framework [86] and the discussion on the selection of PGM as the interpretable model explaining GNNs. A detailed description of PGM-Explainer is provided in Section 5.2. The experimental evaluations of different explainers are reported in Section 5.3. Section 5.4 concludes this chapter.

1 The proofs that the vanilla gradient-based explanation method and GNNExplainer fall into the class of *additive feature attribution methods* are provided in Appendix D.

5.1 Preliminaries

Given a GNN model Φ and a target to be explained t, let $\Phi_t : \mathcal{G} \to \mathcal{K}$ be a prediction to be explained. Here, $\mathcal{G}$ is the set of all possible input graphs of the model and $\mathcal{K}$ is the classification's space. In a node classification, $\Phi(G)$ is the vector of predictions on all nodes of $G \in \mathcal{G}$ and $\Phi_t(G) \equiv \Phi(G)_t$ is the target prediction. For a graph classification, $\Phi(G)$ is the prediction on G and we simply set $\Phi_t(G)$ to be $\Phi(G)$. In GNN, each input graph $G = (V, E)$ with F features on each node is fed into Φ using the $|V| \times F$ features matrix X and the $|V| \times |V|$ adjacency matrix A. The analysis in this chapter considers the black-box setting where the explainers do not have any information on the internal of Φ. Specifically, the explainers are allowed to observe different predictions by performing multiple queries on Φ; however, back-propagation and similar operations based on the model's parameters are not allowed. Regarding the notations, when a graph's component is associated with a random variable, bold notation is used, e.g., a node v is associated with the random variable $\boldsymbol{v}$.

An explanation ζ of Φ_t is normally drawn from a set of possible explanations, called interpretable domain $\mathcal{E}$. The $\mathcal{E}$ explaining GNNs can vary from a subset of edges of G [155] to a subset of entries in X [106]. This chapter adopts an explanation model framework proposed in [86] for neural networks and considers $\mathcal{E}$ to be a family of interpretable models. In this explanation model, given an objective function $R_{\Phi,t} : \mathcal{E} \to \mathbb{R}$ associating each explanation with a score, the explanation can be considered as the solution of the following optimization problem:

$$\zeta^* = \arg\max_{\zeta \in \mathcal{E}} R_{\Phi,t}(\zeta). \tag{5-1}$$

To encourage a compact solution ζ^*, explainers might introduce some constraints on (5-1). Specifically, a general condition $\zeta \in \mathcal{C}$ where $\mathcal{C} \subseteq \mathcal{E}$ is typically used to represent those constraints. For instance, we can promote a more simple model ζ^* by setting $\mathcal{C}$ to be a set of models with a limited number of free parameters.

Selecting an appropriate interpretable domain $\mathcal{E}$ is crucial to the explainer's quality. The first reason is the trade-off in the complexity of $\mathcal{E}$. On one hand, $\mathcal{E}$ must be complex enough to explain the target prediction. On the other hand, $\mathcal{E}$ should be simple so that end-users can interpret the explanation. Intuitively, for each $\zeta \in \mathcal{E}$ to explain Φ_t faithfully, the behavior of ζ must be similar to that of Φ_t. Hence, in order to explain GNNs, $\mathcal{E}$ must be chosen such that it contains models having similar behaviors to that of GNNs' predictions.

Probabilistic Graphical models (PGMs) [73] are statistical models encoding complex distributions over a multi-dimensional space using graph-based representation. In general, the probabilistic graph of a distribution represents the distribution compactly in a factorized form. Knowing the PGM not only allows the distribution to be written down tractably but also provides a simple interpretation of the dependencies of those underlying random variables. As our target of explanation is a complex function Φ_t with high dependencies among input features, approximating Φ_t by linear functions can lead to poor results. In contrast, PGM encodes rich sets of information on the graph's components which can potentially support us in analyzing the contributions of each component toward the examined variable. For instance, PGM is able to tell us certain explained features that can determine the target prediction only under specific realizations of some other features. Such kind of information clearly cannot be obtained from linear models.

Bayesian network [102], a PGM representing the conditional dependencies among variables via a directed acyclic graph, is one of the most well-known PGM due to its intuitive representation. Furthermore, efficient algorithms searching for Bayesian network given sampled data, known as structure learning, have been studied extensively [73, 119]. As such, given target of explanation Φ_t, the proposed PGM explanation is the optimal Bayesian network $\mathcal{B}^*$ of the following optimization:

$$\arg\max_{\mathcal{B}\in\mathcal{E}} R_{\Phi,t}(\mathcal{B}), \quad \text{s.t. } |\mathcal{V}(\mathcal{B})| \le M, \boldsymbol{t} \in \mathcal{V}(\mathcal{B}), \tag{5-2}$$

where $\mathcal{E}$ is the set of all Bayesian networks. $\mathcal{V}(\mathcal{B})$ is the set of random variables in Bayesian network $\mathcal{B}$ and $\boldsymbol{t}$ is the random variable corresponding to the target prediction t. In optimization (5-2), the first constraint ensures that the number of variables in $\mathcal{B}$ is bounded by a given constant

M to encourage a compact solution and the second constraint guarantees the target prediction is included in the explanation.

PGM-Explainer also supports the searching for PGM with the following *no-child* constraint:

$$\mathrm{Ch}_{\mathcal{B}}(\boldsymbol{t}) = \emptyset \tag{5-3}$$

where $\mathrm{Ch}_{\mathcal{B}}(\boldsymbol{t})$ is the set of children of node $\boldsymbol{t}$ in Bayesian network $\mathcal{B}$. We introduce this constraint because, in some cases, a target variable $\boldsymbol{t}$ is more desirable to be a leaf in a PGM explanation. This condition not only lets us answer conditional probability queries on target variables more efficiently but also makes the resulting PGM more intuitive. For instance, in a Bayesian network, the parents of the same child can directly influence the distributions of each other even though there might be no edge among them. This additional constraint allows us to avoid this ambiguity in explaining the target variable. We provide an illustrative example of this *no-child* constraint in Appendix E.

5.2 PGM-Explainer: Probabilistic Graphical Model Explanations for GNNs

An example of a PGM explanation is demonstrated in Fig 5-1. The test approach is adopted from [155] where the input graph is a combination of a Barabási-Albert (BA) graph, a set of motifs and some random edges.[2] Nodes are assigned into four classes based on their roles as shown by different colors in Fig. 5-1a. A ground-truth explanation of a prediction on a node in a motif is all the nodes in the motif. In this example, the target of explanation is the role ”purple” of node E in Fig. 5-1b. Our PGM-Explainer is able to identify all nodes in the motif and constructs a PGM approximating the target prediction (Fig. 5-1c). Different from existing explainers, PGM-Explainer provides statistical information on the contributions of the graph’s components in terms of conditional probabilities. For example, without knowing any information on E’s neighborhood, the PGM explanation approximates a probability of predicting E to be ”purple” is 47.2%. If PGM knew a prediction of node A and its realization, that probability would be increased to 65.8%. This information not only helps users evaluate the contribution of each

2 A scale-free network is a network whose degree distribution follows power law. The Barabási-Albert graph is a scale-free network generated by the Barabási–Albert model [11]

explained feature on the target prediction but also provides intuition on their interactions in constituting that prediction.

PGM-Explainer consists of three major steps: data generation, variable selection, and structure learning, which are summarized in Fig. 5-2. The data generation step is to generate, preprocess, and record a set of input-output pairs, called sampled data, of the prediction to be explained. The variables selection step eliminates unimportant variables from the sampled data to improve the running time and encourage compact explanations. The final step, structure learning, takes the filtered data from the previous step and generates a PGM explanation.

5.2.1 Data Generation

The goal of the data generation step in PGM-Explainer is to generate a set of sampled data $\mathcal{D}_t$ from the target function Φ_t. In the consequence steps, the PGM will be learned from this sampled data $\mathcal{D}_t$. Since the explainer aims to capture the behaviors of Φ_t, especially at the neighborhoods of input graph G, PGM-Explainer first perturbs some features of G to obtain a set of perturbed samples. Specifically, we fix a parameter $p \in (0,1)$ representing a probability that the features in each node is perturbed. For each node v in G, we introduce a random variable s_v indicating whether the features on v are perturbed. The perturbation scheme might vary depending on applications since we want the perturbed graph to be in the vicinity of G. We implement different perturbing algorithms; however, the scheme used in this chapter is simply setting the node features to the mean value among all nodes. For each realization of $\mathbf{s} = \{s_v\}_{v \in V}$, we obtain an induced graph $G(\mathbf{s}) \in \mathcal{G}$. The prediction $\Phi(G(\mathbf{s}))$ on the induced graph is then obtained by feeding $G(\mathbf{s})$ through the GNN.

In a node classification task, for each perturbation, the realization of node variable $\boldsymbol{v} = \{s_v, \mathrm{I}(\Phi(G(\mathbf{s}))_v)\}$ is recorded into $\mathcal{D}_t$, where $\mathrm{I}(.)$ is a function indicating whether the prediction $\Phi(G(\mathbf{s}))_v$ is different significantly from the original prediction $\Phi(G)_v$. Intuitively, $\boldsymbol{v}$ encodes both the influence of features of node v onto the graph's prediction and the influence of the overall changes in the graph to the prediction on v. In our implementations of PGM-Explainer, s_v and $\mathrm{I}(.)$ are stored in binary values and the domain of $\boldsymbol{v}$ is consequently implemented using only two bits. If the GNN has L graph neural network layers, the prediction on t can only be influenced by L-hop neighbors of t. Hence, the explainer only needs to examine the L-hop neighbors of t. Thus, for each realization $\mathbf{s}$, PGM-Explainer only records $\boldsymbol{v}$ for $v \in \text{Ne}_t^G$ where Ne_t^G is L-hop neighbors of t in G. After n samplings of $\mathbf{s}$, PGM-Explainer will obtain a data table $\mathcal{D}_t$ of size $n \times |\text{Ne}_t^G|$, where each entry $\mathcal{D}_{t_{iv}} = \boldsymbol{v}^{(i)}$ is the i^{th} sampling of node random variable $\boldsymbol{v}$.

For a graph classification task, we set the node variable $\boldsymbol{v} = \{s_v\}$ and introduce an additional target variable $\boldsymbol{t} = \{\text{I}(\Phi(G(\mathbf{s})))\}$ into the set of random variables. Furthermore, we set the neighborhood set Ne_t^G to be V since all nodes on the input graph can influence the graph prediction. Therefore, the size of $\mathcal{D}_t$ in this case is $n \times (|V| + 1)$.

5.2.2 Variables Selections

The task of learning PGM $\mathcal{B}$ from $\mathcal{D}_t$ is called structure learning. Unfortunately, finding an optimal PGM from sampled data is intractable [73]. As Ne_t^G might contain thousands of nodes, searching for an optimal Bayesian network is very expensive. This challenge requires us to trim the set of variables to be examined by the structure learning algorithms.

To reduce the number of variables in $\mathcal{D}_t$, we need to identify which variables are important and avoid eliminating them. PGM-Explainer addresses this problem by observing that important variables to target variable $\boldsymbol{t}$ are in the Markov-blanket of $\boldsymbol{t}$. By definition, the Markov-blanket of $\boldsymbol{t}$ in a PGM $\mathcal{P}$, denoted as $\text{MB}_{\mathcal{P}}(\boldsymbol{t})$, is the minimum set of variables such that, given realizations of all variables in the Markov-blanket, $\boldsymbol{t}$ is conditionally independent from all other variables in $\mathcal{P}$.[3] Therefore, under an assumption that there exists a perfect map $\mathcal{B}^*$ for the distribution of random variables in Ne_t^G, analyzing $\text{MB}_{\mathcal{B}^*}(\boldsymbol{t})$ provides us the same statistical information on $\boldsymbol{t}$ as in Ne_t^G.[4] Thus, instead of finding $\mathcal{B}^*$ on Ne_t^G, we can determine $\text{MB}_{\mathcal{B}^*}(\boldsymbol{t})$ and compute PGM $\mathcal{B}$ on $\text{MB}_{\mathcal{B}^*}(\boldsymbol{t})$ as an explanation for Φ_t. Due to the property of Markov-blanket, statistical information of $\boldsymbol{t}$ in $\mathcal{B}$ and $\mathcal{B}*$ are the same.

Markov-blanket $\text{MB}_{\mathcal{B}^*}(\boldsymbol{t})$ can be obtained by the Grow-Shrink (GS) algorithm [88]. However, the number of conditional independent tests in GS can be in an exponential order of the

3 Formal definition of Markov-blanket is provided in Appendix F.2
4 Formal definition of a perfect map is provided in Appendix F.1

number of random variables. Fortunately, in order to generate explanations for GNN's predictions, the explainer does not need to know exactly $\mathrm{MB}_{\mathcal{B}^*}(\boldsymbol{t})$. In fact, knowing any set containing $\mathrm{MB}_{\mathcal{B}^*}(\boldsymbol{t})$ is sufficient for the resulted PGM to contain all the information of the target prediction. In PGM-Explainer, we propose to find a small subset $\boldsymbol{U}(\boldsymbol{t})$ of Ne_t^G which is guaranteed to contain $\mathrm{MB}_{\mathcal{B}^*}(\boldsymbol{t})$. A definition of such $\boldsymbol{U}(\boldsymbol{t})$ and the guarantee that $\mathrm{MB}_{\mathcal{B}^*}(\boldsymbol{t}) \subseteq \boldsymbol{U}(\boldsymbol{t})$ is given in Theorem 5-1:

Theorem 5-1. *Assume there exists a perfect map $\mathcal{B}^*$ of a distribution P on a set of random variables $\boldsymbol{V}$. For any random variable $\boldsymbol{v}$ in $\boldsymbol{V}$, we denote $\boldsymbol{S}(\boldsymbol{v}) = \{\boldsymbol{v}' \in \boldsymbol{V} | \boldsymbol{v}' \not\perp\!\!\!\perp \boldsymbol{v}\}$. Then, for any $\boldsymbol{t} \in \boldsymbol{V}$, we have $\mathrm{MB}_{\mathcal{B}^*}(\boldsymbol{t}) \subseteq \boldsymbol{U}(\boldsymbol{t})$ where $\boldsymbol{U}(\boldsymbol{t}) = \cup_{\boldsymbol{v} \in \boldsymbol{S}(\boldsymbol{t})} \boldsymbol{S}(\boldsymbol{v})$.*

Proof. Let's consider a node $\boldsymbol{v}' \in \mathrm{MB}_{\mathcal{B}^*}(\boldsymbol{t})$. In Bayesian network $\mathcal{B}^*$, the Markov-blanket of $\boldsymbol{t}$ is the union of $\boldsymbol{t}$'s parents, $\boldsymbol{t}$'s children, and $\boldsymbol{t}$'s children's other parents [73]. In other words, at least one of the three following cases must hold:

1. $\boldsymbol{v}' \in \mathrm{Pa}_{\mathcal{B}^*}(\boldsymbol{t})$.
2. $\boldsymbol{v}' \in \mathrm{Ch}_{\mathcal{B}^*}(\boldsymbol{t})$.
3. $\exists \boldsymbol{v}$ such that $\boldsymbol{v}' \in \mathrm{Pa}_{\mathcal{B}^*}(\boldsymbol{v})$ and $\boldsymbol{t} \in \mathrm{Pa}_{\mathcal{B}^*}(\boldsymbol{v})$.

For the first two cases, either $\boldsymbol{v}' \to \boldsymbol{t}$ or $\boldsymbol{t} \to \boldsymbol{v}'$ is an active path of $\mathcal{B}^*$. As $\mathcal{B}^*$ is a perfect map for P, we have $\boldsymbol{v}' \not\perp\!\!\!\perp \boldsymbol{t}$ and $\boldsymbol{v}'$ must be included in $\boldsymbol{S}(\boldsymbol{t})$. Since $\boldsymbol{v} \in \boldsymbol{S}(\boldsymbol{v})$ for all $\boldsymbol{v}$, we have $\boldsymbol{S}(\boldsymbol{t}) \subseteq \cup_{\boldsymbol{v} \in \boldsymbol{S}(\boldsymbol{t})} \boldsymbol{S}(\boldsymbol{v})$ and $\boldsymbol{v}' \in \cup_{\boldsymbol{v} \in \boldsymbol{S}(\boldsymbol{t})} \boldsymbol{S}(\boldsymbol{v})$.

For the last case, as $\boldsymbol{v}' \in \mathrm{Pa}_{\mathcal{B}^*}(\boldsymbol{v})$ and $\boldsymbol{t} \in \mathrm{Pa}_{\mathcal{B}^*}(\boldsymbol{v})$, we have $\boldsymbol{v}' \to \boldsymbol{v}$ and $\boldsymbol{t} \to \boldsymbol{v}$ are active paths of $\mathcal{B}^*$. Thus, $\boldsymbol{v}' \not\perp\!\!\!\perp \boldsymbol{v}$ and $\boldsymbol{v} \not\perp\!\!\!\perp \boldsymbol{t}$. Consequently $\boldsymbol{v}'$ must be included in $\boldsymbol{S}(\boldsymbol{v})$ where $\boldsymbol{v} \in \boldsymbol{S}(\boldsymbol{t})$, i.e. $\boldsymbol{v}' \in \cup_{\boldsymbol{v} \in \boldsymbol{S}(\boldsymbol{t})} \boldsymbol{S}(\boldsymbol{v})$. □

Based on Theorem 5-1, constructing $\boldsymbol{U}(\boldsymbol{t})$ from $\mathcal{D}_t$ is straightforward. PGM-Explainer first computes $O\left(|\mathrm{Ne}_t^G|^2\right)$ independent tests to construct $\boldsymbol{S}(\boldsymbol{v})$ $\forall$, $\boldsymbol{v}$. Then $\boldsymbol{U}(\boldsymbol{t})$ is obtained by combining all elements of $\boldsymbol{S}(\boldsymbol{v})$, $\forall$ $\boldsymbol{v} \in \boldsymbol{S}(\boldsymbol{t})$. In case *no-child* constraint (5-3) is considered, we can solve for a much smaller set $\boldsymbol{U}(\boldsymbol{t})$ containing $\mathrm{MB}_{\mathcal{B}^*}(\boldsymbol{t})$ based on Theorem 5-2:

Theorem 5-2. *Assume there exists a perfect map $\mathcal{B}^*$ of a distribution P on $\boldsymbol{V}$. If node $\boldsymbol{t} \in \boldsymbol{V}$ has no child in $\mathcal{B}^*$,* $\mathrm{MB}_{\mathcal{B}^*}(\boldsymbol{t}) \subseteq \boldsymbol{U}(\boldsymbol{t})$ *where* $\boldsymbol{U}(\boldsymbol{t}) = \boldsymbol{S}(\boldsymbol{t}) \triangleq \{\boldsymbol{v} | \boldsymbol{v} \not\perp\!\!\!\perp \boldsymbol{t}\}$.

Proof. Since $\boldsymbol{t}$ has no child, $\mathrm{MB}_{\mathcal{B}^*}(\boldsymbol{t}) = \mathrm{Pa}_{\mathcal{B}^*}(\boldsymbol{t})$ where $\mathrm{Pa}_{\mathcal{B}^*}(\boldsymbol{t})$ is the set of parents of $\boldsymbol{t}$ in $\mathcal{B}^*$. Hence, for any node $\boldsymbol{v} \in \mathrm{MB}_{\mathcal{B}^*}(\boldsymbol{t})$, $\boldsymbol{v} \in \mathrm{Pa}_{\mathcal{B}^*}(\boldsymbol{t})$ and $\boldsymbol{v} \to \boldsymbol{t}$ is an active path of $\mathcal{B}^*$. As $\mathcal{B}^*$ is a perfect map for P, we have $\boldsymbol{v} \not\perp\!\!\!\perp \boldsymbol{t}$ and $\boldsymbol{v}$ must be included in $\boldsymbol{S}(\boldsymbol{t})$. $\square$

Note that, in this case, we can learn the set $\boldsymbol{U}(\boldsymbol{t})$ using only $O\left(|\mathrm{Ne}_t^G|\right)$ independent tests.

Given $\boldsymbol{U}(\boldsymbol{t})$, the explainer can learn the PGM on top of $\boldsymbol{U}(\boldsymbol{t})$ instead of on $|\mathrm{Ne}_t^G|$ random variables. We want to emphasize that the $\boldsymbol{U}(\boldsymbol{t})$'s construction in both Theorem 5-1 and 5-2 of PGM-Explainer only use pairwise dependence tests, not conditional dependence tests as in conventional algorithms solving for the Markov-blankets. This is a significant difference since a conditional dependence test conditioning on m binary variables further requires 2^m dependence tests. Two variables are declared dependent if any of these dependency tests asserts that the distributions are different. When the number of data samples is limited, the probability of including wrong variables into the Markov-blanket is rapidly increasing with the conditioning set size [88]. Additionally, by using the dependency tests only, we can compute $\boldsymbol{U}(\boldsymbol{v})$ for all $\boldsymbol{v}$ in the PGM. Thus, for node classification tasks, PGM-Explainer is able to generate batch explanations for many target predictions simultaneously.

5.2.3 Structure Learning

The final step of PGM-Explainer is to learn the explanation Bayesian network $\mathcal{B}$ from $\mathcal{D}_t$. We first demonstrate the learning without constraint (5-3) with the following *BIC score*:

$$R_{\Phi,t}(\mathcal{B}) = \mathrm{score}_{BIC}(\mathcal{B} : \mathcal{D}_t[\boldsymbol{U}(\boldsymbol{t})]) = l(\hat{\theta}_{\mathcal{B}} : \mathcal{D}_t[\boldsymbol{U}(\boldsymbol{t})]) - \frac{\log n}{2}\mathrm{Dim}[\mathcal{B}] \tag{5-4}$$

where $\mathcal{D}_t[\boldsymbol{U}(\boldsymbol{t})]$ is data $\mathcal{D}_t$ on variables $\boldsymbol{U}(\boldsymbol{t})$ and $\mathrm{Dim}[\mathcal{B}]$ is the dimension of model $\mathcal{B}$. $\theta_{\mathcal{B}}$ are the parameters of $\mathcal{B}$ and function $l(\theta_{\mathcal{B}} : \mathcal{D}_t[\boldsymbol{U}(\boldsymbol{t})])$ is the log-likelihood between $\mathcal{D}_t[\boldsymbol{U}(\boldsymbol{t})]$ and $\theta_{\mathcal{B}}$, i.e. $l(\theta_{\mathcal{B}} : \mathcal{D}_t[\boldsymbol{U}(\boldsymbol{t})]) = P(\mathcal{D}_t[\boldsymbol{U}(\boldsymbol{t})] | \theta_{\mathcal{B}}, \mathcal{B})$. $\hat{\theta}_{\mathcal{B}}$ in (5-4) is the parameters' value that maximizes the log-likelihood, which is called the maximum likelihood estimator. Given this objective, PGM-Explainer can use exhaustive-search to solve for an optimal PGM. However, we observe that the hill-climbing algorithm [54] also returns good local optimal solutions with a significantly lower running time.

In PGM-Explainer, we chose the *BIC score* objective because this objective is proven to be *consistent* with the data [73]. A scoring function is consistent if, as the number of samples $n \to \infty$, the two following conditions hold: (i) $\mathcal{B}^*$ maximizes the score and (ii) all structures that do not contain the same set of independencies with $\mathcal{B}^*$ have strictly lower scores. The second condition is also known as the *I-equivalent* condition. These two properties imply that *BIC score* asymptotically prefers a structure that exactly fits the dependencies in the data. Consequently, with large enough samplings, Bayesian network $\mathcal{B}$ obtained by maximizing the *BIC score* should reflect the dependencies of variables in $\boldsymbol{U}(\boldsymbol{t})$ and thus provides us the reasons behind the model's decision. In Appendix F.3, we provide a more formal discussion on the *BIC score*.

The pseudo-code of PGM-Explainer without no-child constraint is shown in Alg. 5-1. In summary, first, data $\mathcal{D}_t$ is generated by random perturbations on input features. Then, PGM-Explainer trims down the number of variables using pairwise independence tests. Finally, the PGM is learned using *BIC score* with a hill-climbing algorithm.

To impose the *no-child* constraint (5-3), we need to modify the structure learning step to ensure that solutions remain consistent with the data, as shown in Alg. 5-2. Instead of maximizing *BIC score* on $\mathcal{D}_t[\boldsymbol{U}(\boldsymbol{t})]$, PGM-Explainer computes an optimal PGM $\mathcal{B}'$ on $\mathcal{D}'_t = \mathcal{D}_t[\boldsymbol{U}(\boldsymbol{t}) \setminus \{\boldsymbol{t}\}]$. Then, the set of $\boldsymbol{t}$'s parents is obtained by iteratively removing variables from $\boldsymbol{U}(\boldsymbol{t}) \setminus \{\boldsymbol{t}\}$. This can be done in a similar manner as the shrinking step in GS [88]. After that, PGM-Explainer includes $\boldsymbol{t}$ back into $\mathcal{B}'$ and adds directed edges from all parents to t to get the final explanation. The following Theorem shows that, the obtained PGM $\hat{\mathcal{B}}$ is *I-equivalence* to $\mathcal{B}^*$.

Theorem 5-3. *Assume there exists a perfect map $\mathcal{B}^*$ of a distribution P. For a variable $\boldsymbol{t}$ having no child in $\mathcal{B}^*$, $\mathcal{D}'_t$ is the set of n sampling data from P without the data for $\boldsymbol{t}$, $\mathrm{Pa}_{\mathcal{B}^*}(\boldsymbol{t})$ is the set of parents of $\boldsymbol{t}$ in $\mathcal{B}^*$ and $\hat{\mathcal{B}}'$ is an optimal solution of $\max_{\mathcal{B}'} \mathrm{score}_{BIC}(\mathcal{B}' : \mathcal{D}'_t)$. As $n \to \infty$, a Bayesian network $\hat{\mathcal{B}}$ obtaining by adding node $\boldsymbol{t}$ to $\hat{\mathcal{B}}'$ and adding edges $(\boldsymbol{v}, \boldsymbol{t})$ for all $\boldsymbol{v} \in \mathrm{Pa}_{\mathcal{B}^*}(\boldsymbol{t})$ is I-equivalence to $\mathcal{B}^*$ with probability 1.*

Proof. From proposition 18.1 [73], we have the log-likelihood between a data $\mathcal{D}$ on random variables V and the model $\mathcal{B}$ with parameters $\hat{\theta}_{\mathcal{B}}$, $l(\hat{\theta}_{\mathcal{B}} : \mathcal{D})$, can be rewritten as follows

$$l(\hat{\theta}_{\mathcal{B}} : \mathcal{D}) = n \sum_{\boldsymbol{v} \in V} \boldsymbol{I}_{\hat{P}}(\boldsymbol{v}; \mathrm{Pa}_{\mathcal{B}}(\boldsymbol{v})) - n \sum_{\boldsymbol{v} \in V} \mathrm{H}_{\hat{P}}(\boldsymbol{v}) \tag{5-5}$$

where $\hat{P}$ is the empirical distribution.

We now consider the first case where the output $\hat{\mathcal{B}}$ of Algorithm 5-2 implies an independence assumption that $\mathcal{B}^*$ does not support. Thus, the set of independence assumptions of $\hat{\mathcal{B}}$ is not contained in that set in P and we say, by the definition of *I-map* [73], $\hat{\mathcal{B}}$ is not an I-map of P. Therefore, the log-likelihood $l(\hat{\theta}_{\hat{\mathcal{B}}} : \mathcal{D}_t)$ is less than $l(\hat{\theta}_{\mathcal{B}^*} : \mathcal{D}_t)$ and we have

$$\sum_{\boldsymbol{v} \in V} \boldsymbol{I}_P(\boldsymbol{v}; \mathrm{Pa}_{\mathcal{B}^*}(\boldsymbol{v})) > \sum_{\boldsymbol{v} \in V} \boldsymbol{I}_P(\boldsymbol{v}; \mathrm{Pa}_{\hat{\mathcal{B}}}(\boldsymbol{v})) \tag{5-6}$$

where V is the set of nodes of $\mathcal{B}^*$.

Since the sets of parents of $\boldsymbol{t}$ in $\mathcal{B}^*$ and that set in $\hat{\mathcal{B}}$ are the same, we have

$$\sum_{\boldsymbol{v} \in V \setminus \{\boldsymbol{t}\}} \boldsymbol{I}_P(\boldsymbol{v}; \mathrm{Pa}_{\mathcal{B}^*}(\boldsymbol{v})) + \boldsymbol{I}_P(\boldsymbol{t}; \mathrm{Pa}_{\mathcal{B}^*}(\boldsymbol{t})) > \sum_{\boldsymbol{v} \in V \setminus \{\boldsymbol{t}\}} \boldsymbol{I}_P(\boldsymbol{v}; \mathrm{Pa}_{\hat{\mathcal{B}}}(\boldsymbol{v})) + \boldsymbol{I}_P(\boldsymbol{t}; \mathrm{Pa}_{\hat{\mathcal{B}}}(\boldsymbol{t})) \tag{5-7}$$

$$\Rightarrow \sum_{\boldsymbol{v} \in V \setminus \{\boldsymbol{t}\}} \boldsymbol{I}_P(\boldsymbol{v}; \mathrm{Pa}_{\mathcal{B}^*}(\boldsymbol{v})) > \sum_{\boldsymbol{v} \in V \setminus \{\boldsymbol{t}\}} \boldsymbol{I}_P(\boldsymbol{v}; \mathrm{Pa}_{\hat{\mathcal{B}}}(\boldsymbol{v})) \tag{5-8}$$

We denote $\mathcal{B}'^*$ the Bayesian network obtained by removing node $\boldsymbol{t}$ and the related edges from $\mathcal{B}^*$. Since $\boldsymbol{t}$ has no child in $\mathcal{B}^*$, $\mathrm{Pa}_{\mathcal{B}^*}(\boldsymbol{v}) = \mathrm{Pa}_{\mathcal{B}'^*}(\boldsymbol{v})$ for all $\boldsymbol{v} \in V \setminus \{\boldsymbol{t}\}$. Thus, we have

$$\sum_{\boldsymbol{v} \in V \setminus \{\boldsymbol{t}\}} \boldsymbol{I}_P(\boldsymbol{v}; \mathrm{Pa}_{\mathcal{B}^*}(\boldsymbol{v})) = \sum_{\boldsymbol{v} \in V \setminus \{\boldsymbol{t}\}} \boldsymbol{I}_P(\boldsymbol{v}; \mathrm{Pa}_{\mathcal{B}'^*}(\boldsymbol{v})) \tag{5-9}$$

Similarly, as $\boldsymbol{t}$ has no child in $\hat{\mathcal{B}}$, $\mathrm{Pa}_{\hat{\mathcal{B}}}(\boldsymbol{v}) = \mathrm{Pa}_{\hat{\mathcal{B}}'}(\boldsymbol{v})$ for all $\boldsymbol{v} \in V \in V \setminus \{\boldsymbol{t}\}$ and the following holds

$$\sum_{\boldsymbol{v} \in V \setminus \{\boldsymbol{t}\}} \boldsymbol{I}_P(\boldsymbol{v}; \mathrm{Pa}_{\hat{\mathcal{B}}}(\boldsymbol{v})) = \sum_{\boldsymbol{v} \in V \setminus \{\boldsymbol{t}\}} \boldsymbol{I}_P(\boldsymbol{v}; \mathrm{Pa}_{\hat{\mathcal{B}}'}(\boldsymbol{v})) \tag{5-10}$$

Combining (5-8), (5-9) and (5-10), we have:

$$\sum_{\boldsymbol{v} \in V \setminus \{\boldsymbol{t}\}} \boldsymbol{I}_P(\boldsymbol{v}; \mathrm{Pa}_{\mathcal{B}'^*}(\boldsymbol{v})) > \sum_{\boldsymbol{v} \in V \setminus \{\boldsymbol{t}\}} \boldsymbol{I}_P(\boldsymbol{v}; \mathrm{Pa}_{\hat{\mathcal{B}}'}(\boldsymbol{v})) \tag{5-11}$$

This leads us to the following results on the *BIC score* of $\mathcal{B}'^{*}$ and $\hat{\mathcal{B}}'$ as $n \to \infty$:

$$\text{score}_{BIC}(\mathcal{B}'^{*} : \mathcal{D}'_t) - \text{score}_{BIC}(\hat{\mathcal{B}}' : \mathcal{D}'_t) \tag{5-12}$$
$$= l(\hat{\theta}_{\mathcal{B}'^{*}} : \mathcal{D}'_t) - l(\hat{\theta}_{\hat{\mathcal{B}}'} : \mathcal{D}'_t) - \frac{\log n}{2}\left(\text{Dim}[\mathcal{B}'^{*}] - \text{Dim}[\hat{\mathcal{B}}']\right) \tag{5-13}$$
$$= n\left(\sum_{\boldsymbol{v}\in V\setminus\{\boldsymbol{t}\}} \boldsymbol{I}_{\hat{P}}(\boldsymbol{v}; \text{Pa}_{\mathcal{B}'^{*}}(\boldsymbol{v})) - \sum_{\boldsymbol{v}\in V\setminus\{\boldsymbol{t}\}} \boldsymbol{I}_{\hat{P}}(\boldsymbol{v}; \text{Pa}_{\hat{\mathcal{B}}'}(\boldsymbol{v}))\right) - \frac{\log n}{2}\left(\text{Dim}[\mathcal{B}'^{*}] - \text{Dim}[\hat{\mathcal{B}}']\right) \tag{5-14}$$
$$\approx n\Delta - \frac{\log n}{2}\left(\text{Dim}[\mathcal{B}'^{*}] - \text{Dim}[\hat{\mathcal{B}}']\right) \tag{5-15}$$

where $\Delta = \sum_{\boldsymbol{v}\in V\setminus\{\boldsymbol{t}\}} \boldsymbol{I}_P(\boldsymbol{v}; \text{Pa}_{\mathcal{B}'^{*}}(\boldsymbol{v})) - \sum_{\boldsymbol{v}\in V\setminus\{\boldsymbol{t}\}} \boldsymbol{I}_P(\boldsymbol{v}; \text{Pa}_{\hat{\mathcal{B}}'}(\boldsymbol{v}))$. The last approximation (5-15) is from the fact that, as $n \to \infty$, the empirical distribution $\hat{P}$ converges to P. Since $\Delta > 0$ by (5-11) the expression in (5-15) approaches positive infinity when $n \to \infty$ and we have $\text{score}_{BIC}(\mathcal{B}'^{*} : \mathcal{D}'_t) > \text{score}_{BIC}(\hat{\mathcal{B}}' : \mathcal{D}'_t)$. This is a contradiction to the fact that $\hat{\mathcal{B}}'$ maximizes $\text{score}_{BIC}(\mathcal{B}' : \mathcal{D}'_t)$.

We now consider the remaining case where the independence assumptions in $\hat{\mathcal{B}}$ contain all the independence assumptions in $\mathcal{B}^{*}$, and $\mathcal{B}^{*}$ contains an independence assumption that $\hat{\mathcal{B}}$ does not. Thus, $\hat{\mathcal{B}}$ is able to represent any distributions that $\mathcal{B}^{*}$ can. As $\hat{P}$ converges to P when $n \to \infty$, we have the log-likelihood of the two networks are equal:

$$l(\hat{\theta}_{\hat{\mathcal{B}}} : \mathcal{D}_t) = l(\hat{\theta}_{\mathcal{B}^{*}} : \mathcal{D}_t) \tag{5-16}$$

Using similar tricks as in (5-8), (5-9) and (5-10) for the previous case, we will obtain

$$\sum_{\boldsymbol{v}\in V\setminus\{\boldsymbol{t}\}} \boldsymbol{I}_P(\boldsymbol{v}; \text{Pa}_{\mathcal{B}'^{*}}(\boldsymbol{v})) = \sum_{\boldsymbol{v}\in V\setminus\{\boldsymbol{t}\}} \boldsymbol{I}_P(\boldsymbol{v}; \text{Pa}_{\hat{\mathcal{B}}'}(\boldsymbol{v})) \tag{5-17}$$

Now, the difference between the *BIC scores* of $\mathcal{B}'^{*}$ and $\hat{\mathcal{B}}'$ as $n \to \infty$ is

$$\text{score}_{BIC}(\mathcal{B}'^{*} : \mathcal{D}'_t) - \text{score}_{BIC}(\hat{\mathcal{B}}' : \mathcal{D}'_t) \tag{5-18}$$
$$= l(\hat{\theta}_{\mathcal{B}'^{*}} : \mathcal{D}'_t) - l(\hat{\theta}_{\hat{\mathcal{B}}'} : \mathcal{D}'_t) - \frac{\log n}{2}\left(\text{Dim}[\mathcal{B}'^{*}] - \text{Dim}[\hat{\mathcal{B}}']\right) = \frac{\log n}{2}\left(\text{Dim}[\hat{\mathcal{B}}'] - \text{Dim}[\mathcal{B}'^{*}]\right) \tag{5-19}$$

As $\hat{\mathcal{B}}$ has less independence assumptions than $\mathcal{B}^{*}$, $\text{Dim}[\hat{\mathcal{B}}] - \text{Dim}[\mathcal{B}^{*}] > 0$. Furthermore, the reduction in number of parameters from $\hat{\mathcal{B}}$ to $\hat{\mathcal{B}}'$ is the same as the reduction from $\mathcal{B}^{*}$ to $\mathcal{B}'^{*}$

(remove one node and the same set of edges). Thus we have $\text{Dim}[\hat{\mathcal{B}}'] - \text{Dim}[\mathcal{B}'^*] > 0$. This constitutes a contradiction in the assumption that $\hat{\mathcal{B}}'$ maximizes $\text{score}_{BIC}(\mathcal{B}' : \mathcal{D}'_t)$. $\square$

Note that if we run Alg. 5-2 on a target $\boldsymbol{t}$ that has children in the optimal PGM $\mathcal{B}^*$, we will obtain a PGM $\mathcal{B}$ that contains no independence statement that is not in $\mathcal{B}^*$. Furthermore, the Markov-blanket of $\boldsymbol{t}$ in $\mathcal{B}^*$ is the set of parents of $\boldsymbol{t}$ in $\mathcal{B}$. With this, we have finalized the description of PGM-Explainer with *no-child* constraint.

5.3 Experiments

This section provides our experiments, comparing the performance of PGM-Explainer to that of existing explanation methods for GNNs, including GNNExplainer [155] and our implementation of the extension of SHapley Additive exPlanations (SHAP) [86] to GNNs. We select SHAP to represent the gradient-based methods because of two reasons. First, source codes of gradient-based methods for GNNs are either unavailable or limited to specific models/applications. Second, SHAP is an *additive feature attribution methods*, unifying explanation methods for conventional neural networks [86]. By comparing PGM-Explainer with SHAP, we aim to demonstrate the drawbacks of the linear-independence assumption of explained features in explaining GNN's predictions. We also show that the vanilla gradient-based explanation method and GNNExplainer can be considered as *additive feature attribution methods* in Appendix D. Our source code can be found at [141].

5.3.1 Datasets and Experiment Settings

We consider six synthetic datasets, detailed in Table 5-1, for the synthetic node-classification task. In particular, we reuse the source code of [155] as we want to evaluate explainers in the same settings. In these datasets, each input graph is a combination of a base graph and a set of motifs. The ground-truth label of each node on a motif is determined based on its role in the motif. As the labels are determined based on the motif's structure, the explanation for the role's prediction of a node is the nodes in the same motif. Thus, the ground-truth explanation in these datasets is the nodes in the same motif as the target. Since the ground-truth explanations are clear, we use accuracy as an evaluation metric.

For the real-world node-classification task, we use the *Trust weighted signed networks* Bitcoin-Alpha and Bitcoin-OTC datasets [75]. These are networks of 3783 and 5881 accounts trading Bitcoin on platforms called Bitcoin-Alpha and Bitcoin-OTC respectively. In each network, members rate other members on a scale of -10 (total distrust) to +10 (total trust). We label each account *trustworthy* or *not-trustworthy* based on those ratings. For each account, its features encode the account's outgoing information such as the average rate made by that account or the normalized number of votes that account made. Since each account is rated by many others, we do not know exactly which neighbors' information is exploited by the GNN to make its predictions, we evaluate explanations based on precision instead of accuracy. Any accounts in an explanation that do not rate the target or rate the target negatively are counted as ”wrong accounts”. The precision of an explanation is computed by dividing the ”not wrong accounts” by the total number of accounts in that explanation.

For the real-world graph classification task, we use MNIST SuperPixel-Graph dataset [34] of which each sample is a graph corresponding to an image sample in the hand-written digit MNIST dataset [79]. In this dataset, the original MNIST images are converted to graphs using super-pixels, which represent small regions of homogeneous intensity in images. In a given graph, each node represents a super-pixel in the original image while the node's features are the intensity and locations of the super-pixel (see Fig. 5-3(a) for examples). As the ground-truth explanations for this experiment are not available, we use human-subjective test to evaluate the results of explainers.

For each dataset, we use Adam optimizer [70] to train a single GNN for the corresponding task. For the synthetic datasets, the Bitcoin-Alpha dataset and Bitcoin-OTC dataset, we use the GNN models provided by [155]. All models have 3 graph layers with the numbers of parameters between 1102 and 1548. The train/validation/test split is $80/10/10\%$. The models are trained for 1000 epochs with a learning rate of 0.001. For the MNIST SuperPixel-Graph dataset, the dataset is divided into 55K/5K/10K. We use the Graph Convolutional Network (GCN) model [71] implemented by [34] due to its popularity and efficiency in training. The model has 101365 parameters and converges at epoch 188 with learning rate is automatically chosen between 10^{-3} and 10^{-4}. The models' test-set accuracy and the number of samples n used by PGM-Explainer to generate the explanations are reported in Table 5-2.

5.3.2 Results on Synthetic Datasets

Table 5-3 shows the accuracy of the explanations generated by different explainers. Here the explanations are generated for all nodes in the motifs of the input graph. All explainers are programmed to return the top k important nodes to the target prediction where k is the number of nodes in the target's motif. For GNNExplainer, the accuracy we obtained using its default implementation provided in [153] is lower than that reported in the corresponding paper [155]. Perhaps the tuning parameters of GNNExplainer used in [155] are not available to us via [153]. To be consistent, we report the original results in black and our results in blue for GNNExplainer.

As can be seen, PGM-Explainer outperforms other explanation methods, especially on datasets 2 and 6. We specifically designed dataset 6 so that the role of a node on a motif cannot be determined by knowing only one of its neighbors. Furthermore, for some nodes, such as node D and E in Fig. 5-1b, the roles can only be determined using almost all 2-hop neighbors. To achieve high predictive performance on this dataset, the GNN must integrate network features in a non-linear manner. Specifically, observing only red node A does not give the GNN much information on the target. Knowing one of two blue nodes B and C does not provide much information for the prediction either. However, the knowledge of all those three nodes can help the GNN fully determine the role of E. To faithfully explain the GNN, an explainer must be able to capture these non-linear behaviors.

For some synthetic datasets, we observe that GNNs can determine the target node's role without knowing all nodes on the target motif. Thus allowing explainers to return less than k nodes can improve the overall precision. PGM-Explainer can easily make this decision by performing dependence tests on those k nodes. We report the precision in parentheses for PGM-Explainer.

5.3.3 Results on Real-world Datasets

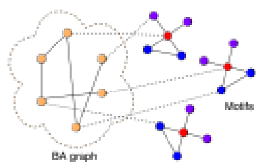
The precision of nodes in explanations of each explainer on Trust weighted signed networks datasets are reported in Table 5-4. Here we test the total number of accounts in each explanation at 3, 4, and 5. Our results show that PGM-Explainer achieves significantly higher precision than other methods in these experiments.

Some example explanations for the MNIST SuperPixel-Graph dataset are reported in Fig 5-3. Here we do not have the results of GNNExplainer because the test model implemented by [34] for this dataset do not support the input of fractional adjacency matrix, which is a requirement for GNNExplainer to work. Instead, we compare PGM-Explainer with the SHAP extension for GNN and GRAD, a simple gradient approach. In this experiment, we do not restrict the number of nodes returned by PGM-Explainer. In fact, the nodes are included as long as they are in the Markov-blanket of the target variable. From that, we obtain explanations containing either $2, 3$, or 4 nodes with an average of 3.08. Since other methods do not have an equivalent mechanism for selecting features, we set their number of returned nodes to 3. In GRAD, the top-3 nodes are chosen based on the sum gradients of the GNN's loss function with respect to the associated node features.

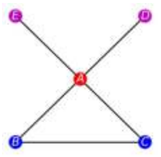
In our human-subjective test, we randomly chose 48 graphs whose predictions made by the GNN were correct. Then we use PGM-Explainer, SHAP, and GRAD to generate the corresponding explanations. Two examples of these explanations are provided in Fig 5-3 where nodes returned in the explanation are highlighted in red boxes. Ten end-users were asked to give a score on a scale of 0 to 10 for each explanation based on their thoughts on the importance of the highlighted nodes to the GNN prediction. From the distribution of the scores in Fig. 5-4, the explanations of PGM-Explainer are evaluated much higher than those of other methods. This means the nodes in the explanations generated by PGM-Explainer are more intuitive and might be more important to the GNN's predictions than those of other explanation methods.

5.4 Conclusion

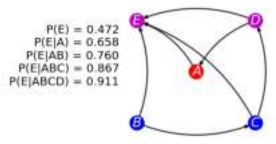
In this chapter, we propose PGM-Explainer, an explanation method faithfully explaining the predictions of any GNN in an interpretable manner. By approximating the target prediction with a graphical model, PGM-Explainer is able to demonstrate the non-linear contributions of explained features toward the prediction. Our experiments not only show the high accuracy and precision of PGM-Explainer but also imply that PGM explanations are favored by end-users. Although we only adopt Bayesian networks as interpretable models, our formulations of PGM-Explainer support the exploration of other graphical models such as Markov networks and Dependency networks. For future research, we would like to analyze the impact of different objective functions and structure learning methods on explainers' quality and running time.



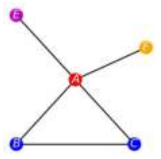
(a) Input graph.



(b) Motif containing E.



(c) PGM-Explainer.



(d) GNNExplainer.

Figure 5-1. Example of PGM-explanation of a GNN's prediction. **(a)** An input graph of the GNN model: a combination of a 300-node BA graph, 80 5-node motifs, and some random edges. The labels of nodes are assigned based on their roles in the graph (shown in color) **(b)** A motif containing the target prediction to be explained. We aim to explain the prediction of the role of node E. **(c)** A PGM-explanation in the form of a Bayesian network. This Bayesian network estimates the probability that node E has the predicted role given a realization of other nodes. **(d)** An explanation of GNNExplainer, which mistakenly includes node F in the BA graph.

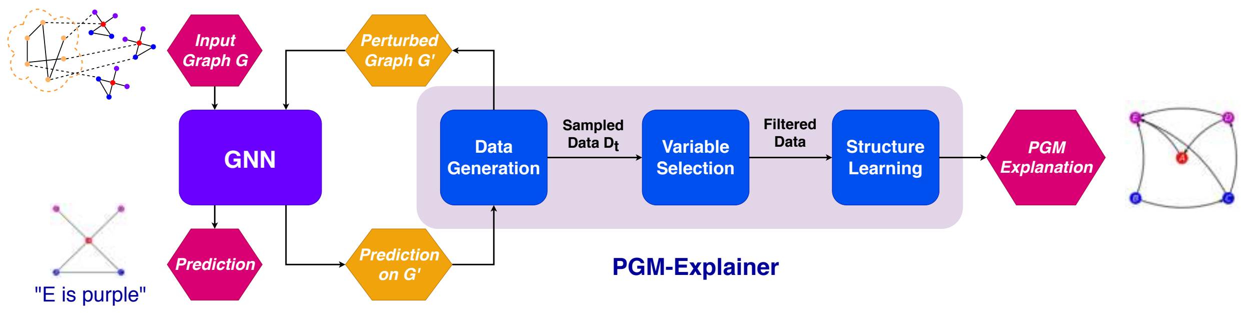


Figure 5-2. The architecture of PGM-Explainer. Given input graph G and a prediction to be explained, PGM-Explainer generates perturbed graphs and records GNN's predictions on those graphs in the data generation step. The variable selection step eliminates unimportant explained features in this data and forwards the filtered data. Finally, the PGM is generated in the structure learning step.

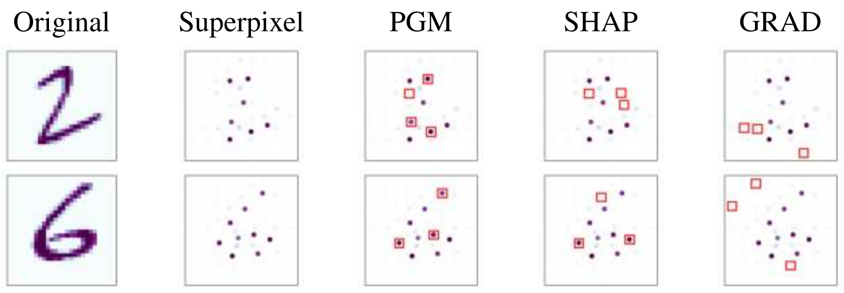


Figure 5-3. Examples and distributions of scores of explanations generated by different explainers on the MNIST SuperPixel-Graph dataset. The nodes returned in each explanation are highlighted in red boxes.

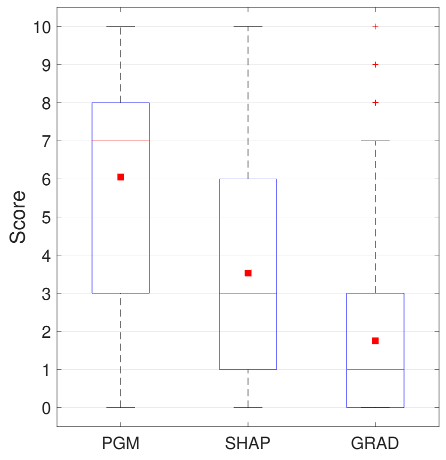


Figure 5-4. The distribution of scores given by end-users on the explanations. The red dots and red bars are the average and median values respectively.

Table 5-1. Parameters of synthetic datasets.

Dataset	Base	Motifs	Node's Features
Syn 1	300-node BA graph	80 5-node house-shaped motif	Constant
Syn 2	350-node BA graph	100 5-node house-shaped motif	Generated from Labels
Syn 3	300-node BA graph	80 9-node grid-shaped motif	Constant
Syn 4	Tree with height 8	60 6-node cycle-shaped motif	Constant
Syn 5	Tree with height 8	80 9-node grid-shaped motif	Constant
Syn 6	300-node BA graph	80 5-node bottle-shaped motif	Constant

Table 5-2. Models' accuracy and number of sampled data used by PGM-Explainer

	Node Classification								Graph Classification
Experiment	Syn 1	Syn 2	Syn 3	Syn 4	Syn 5	Syn 6	Bitcoin-alpha	Bitcoin-OTC	GCN-MNIST
Accuracy	97.9	85.4	100.0	99.4	89.1	99.3	93.9	89.5	90.4
No. Samples	800	800	800	1600	4000	800	1000	1000	400

Table 5-3. Accuracy of Explainers on Synthetic Datasets.

Explainer	Syn 1	Syn 2	Syn 3	Syn 4	Syn 5	Syn 6
SHAP	0.947	0.741	0.872	0.884	0.641	0.741
GNNExplainer	0.925 - 0.729	0.836 - 0.750	0.741	0.948 - 0.862	0.875 - 0.842	0.612
PGM-Explainer	0.965	0.926	0.885 (0.942)	0.954 (0.968)	0.878 (0.892)	0.953

Table 5-4. Precision of Explainers on Trust Signed Networks datasets

Explainer	Bitcoin-alpha			Bitcoin-OTC		
	top 3	top 4	top 5	top 3	top 4	top 5
SHAP	0.537	0.498	0.465	0.607	0.587	0.566
GNNExplainer	0.375	0.332	0.307	0.385	0.338	0.312
PGM-Explainer	0.873	0.857	0.848	0.833	0.817	0.808

```
Input: Model $\Phi$, input graph $G$, target $t$, size constraint $M$, and sampling size $n$.
Output: A Bayesian network $\mathcal{B}_t$ explains prediction $\Phi(G)_t$.

    Step 1: Data generation
 1: $L =$ the number of GNN layers of $\Phi$.
 2: Get the $L$-hop neighbor graph $\mathrm{Ne}_t^G$
 3: Initiate an empty data $\mathcal{D}_t$
 4: for $i = 1$ to $n$ do
 5:     Generate random $\boldsymbol{s} \in \{0,1\}^{|\mathrm{Ne}_t^G|}$
 6:     Compute realization graph $G(\boldsymbol{s})$
 7:     Compute prediction $\Phi(G(\boldsymbol{s}))$
 8:     For each $v \in \mathrm{Ne}_t^G$, compute $\boldsymbol{v} = \{s_v, \mathrm{I}(\Phi(G(\mathbf{s}))_v)\}$
 9:     Record $\boldsymbol{v}$ into the entry of node $v$ in data $\mathcal{D}_t$.
10: end for

    Step 2: Generate $\boldsymbol{U}(\boldsymbol{t})$
11: Initiate an empty set $\boldsymbol{U}(\boldsymbol{t})$
12: For each $v \in \mathrm{Ne}_t^G$, if $\boldsymbol{v} \not\perp\!\!\!\perp \boldsymbol{t}$, add $v$ to $S(\boldsymbol{t})$.
13: For each $v \in S(\boldsymbol{t})$:
14:      For each $v' \in \mathrm{Ne}_t^G \setminus \{v\}$ if $\boldsymbol{v}' \not\perp\!\!\!\perp \boldsymbol{v}$, add $v'$ to $U(\boldsymbol{t})$.
15: Rank the nodes in $\boldsymbol{U}(\boldsymbol{t})$ based on their dependencies with $\boldsymbol{t}$.
16: Keep the top $M$ dependent variables in $\boldsymbol{U}(\boldsymbol{t})$.

    Step 3: Structure Learning - Generate the explanation $\hat{\mathcal{B}}$:
17: $\hat{\mathcal{B}} = \arg\max_{\mathcal{B}} \mathrm{score}_{BIC}(\mathcal{B} : \mathcal{D}_t[\boldsymbol{U}(\boldsymbol{t})])$
18: Return $\hat{\mathcal{B}}$.
```

Algorithm 5-1. Generating PGM explanation for GNN without no-child constraint

Input: Model Φ, input graph G, target t, size constraint M, and sampling size n.
Output: A Bayesian network $\mathcal{B}_t$ explains prediction $\Phi(G)_t$.

```
    Step 1: Data generation
 1: $L =$ the number of GNN layers of $\Phi$.
 2: Get the $L$-hop neighbor graph $\mathrm{Ne}_t^G$
 3: Initiate an empty data $\mathcal{D}_t$
 4: for $i = 1$ to $n$ do
 5:     Generate random $\boldsymbol{s} \in \{0,1\}^{|\mathrm{Ne}_t^G|}$
 6:     Compute realization graph $G(\boldsymbol{s})$
 7:     Compute prediction $\Phi(G(\boldsymbol{s}))$
 8:     For each $v \in \mathrm{Ne}_t^G$, compute $\boldsymbol{v} = \{s_v, \mathrm{I}(\Phi(G(\mathbf{s}))_v)\}$
 9:     Record $\boldsymbol{v}$ into the entry of node $v$ in data $\mathcal{D}_t$.
10: end for

    Step 2: Generate $\boldsymbol{U}(\boldsymbol{t})$
11: Initiate an empty set $\boldsymbol{U}(\boldsymbol{t})$
12: For each $v \in \mathrm{Ne}_t^G$, if $\boldsymbol{v} \not\perp\!\!\!\perp \boldsymbol{t}$, add $v$ to $\boldsymbol{U}(\boldsymbol{t})$.
13: Rank the nodes in $\boldsymbol{U}(\boldsymbol{t})$ based on their dependencies with $\boldsymbol{t}$.
14: Keep the top $M$ dependent variables in $\boldsymbol{U}(\boldsymbol{t})$.

    Step 3: Structure Learning - Generate the explanation $\hat{\mathcal{B}}$:
15: $\hat{\mathcal{B}}' = \arg\max_{\mathcal{B}'} \mathrm{score}_{BIC}(\mathcal{B}' : \mathcal{D}_t[\boldsymbol{U}(\boldsymbol{t}) \setminus \{\boldsymbol{t}\}])$
16: Adding $\boldsymbol{t}$ to $\hat{\mathcal{B}}'$ and obtain $\hat{\mathcal{B}}$.
17: Initiate set $\mathrm{Pa}(\boldsymbol{t}) = \boldsymbol{U}(\boldsymbol{t})$
18: While $\exists \boldsymbol{v} \in \mathrm{Pa}(\boldsymbol{t})$ such that $\boldsymbol{v} \perp\!\!\!\perp \boldsymbol{t} | \mathrm{Pa}(\boldsymbol{t}) - \{\boldsymbol{v}\}$, remove $\boldsymbol{v}$ from $\mathrm{Pa}(\boldsymbol{t})$.
19: For all $\boldsymbol{v} \in \mathrm{Pa}(\boldsymbol{t})$, add edge $(\boldsymbol{v}, \boldsymbol{t})$ to $\hat{\mathcal{B}}$.
20: Return $\hat{\mathcal{B}}$.
```

Algorithm 5-2. Generating PGM explanation for GNN with no-child constraint

CHAPTER 6
UNEXPLAINABLE AI: ON THE LIMIT OF EXPLAINING BLACK-BOX TEMPORAL GRAPH NEURAL NETWORKS

Graph Neural Networks have been achieving successful performance in many practical graph-related problems including social networks, citation networks, and biological networks [156, 160, 163]. Various architectures with elegant designs and competitive performance have been introduced in recently [71, 55, 137]. Along those works, a notable branch of GNNs is developed to integrate temporal information into the graph structure, called Temporal Graph Neural Networks [162, 147, 91]. This variant has shown promising results in domains where the data has strong time correlations such as transportation and weather forecasts.

Since GNNs and TGNNs inherit the black-box nature of neural networks, interpreting their predictions remains non-trivial as internal information about the models is not available. In response, many explanation methods, called explainers, have been introduced to explain local predictions of GNNs [155, 33, 140, 61]. These methods generally rely on the model's responses to some perturbations of the input to find the explanations. While the approach has shown many heuristic successes, there is little theoretical result that follows. In particular, is there any information on the model's internal behavior that a given method of perturbation cannot uncover? Addressing this question will help us design better explainers for variants of GNNs, such as TGNNs and many other architectures to come. More importantly, analyzing the limits of explaining methods also helps prevent false claims and incorrect inferences from the explanations.

Our work focuses on the limit of perturbation-based explanation methods when applying to TGNN, i.e. what information cannot be revealed by some given class of popular perturbations. The classes are categorized based on the input's features that they perturb: *node-only*, *edge-only*, and *node-and-edge*. We introduce a proof structure, called *Unidentifiable Proof*, through which the limit of perturbation methods can be formalized and examined. For each class of explainers, we identify a training task (Fig. 6-1) and construct some models such that there is no method in the class that can identify the internal dynamics of those models when generating predictions. Specifically, given a constant K depending on the model's parameters, we show:

1. Node-perturbation methods bounded by K cannot identify the path carrying out the message passing. (Fig. 6-1a).
2. Edge-perturbation methods cannot identify all nodes contributing to a max aggregation. (Fig. 6-1b).
3. Node-and-Edge-perturbation methods bounded by K cannot identify which nodes carry out the temporal aggregation. (Fig. 6-1c).

In practice, K is the result of training and can be arbitrarily large. Thus, our analysis is relevant and applicable to many practical scenarios as large perturbations are often weighted lightly due to the notion of locality [111]. While most of our results are applicable to GNN, we focus on TGNN due to its lack of study. Another reason is, as TGNNs add the temporal dimension to GNNs, it introduces a corresponding temporal dimension to the explaining problem. We find studying this temporal aspect novel and interesting by itself.

Sect. 6.1 and Sect. 6.2 discuss the related works and preliminaries, respectively. Our proposed Unidentifiable Proof and some related notions are introduced in Sect. 6.3. Sects. 6.4, 6.5 and 6.6 formally describe and prove the type of information that Node-perturbation, Edge-perturbation, and Node-and-Edge-perturbation cannot identify. Sect. 6.7 provides synthetic and real-world experiments showing the impact of perturbation schemes on the explaining tasks. Sect. 6.8 concludes the chapter with discussions on the theoretical implications and practical aspects of our results.

6.1 Related Works

To our knowledge, there is currently no work theoretically studying the limits of explanations for GNNs or TGNNs. Even though many experiments evaluating explanation methods have been conducted [118, 5], there exist many pitfalls and challenges in those evaluations as the ground-truth explanations are often unavailable [39]. Furthermore, with the increasing number of datasets, model architectures, and explanation methods, conducting comprehensive evaluations is becoming much more challenging, especially for black-box methods of which the computation complexity is significantly higher than that of white-box methods [5]. Our work is directly related to black-box perturbation-based explaining methods for GNNs, including GNNExplainer [155], PGExplainer [87], GraphLIME [61], and some others [140, 161, 33, 45]. Table 6-1 summarizes the target of perturbation conducted by those explainers and the scope of results in each section of this paper.

6.2 Preliminaries

We now introduce some preliminaries and notations that are commonly used in the research of GNNs and the explaining problem. We also briefly introduce Dynamic Bayesian Networks, which we use in our Unidentifiable Proofs.

For all models studied in this work, their inputs are defined on a graph $G = (V, E)$, where V is the set of nodes and E is the set edges. The inputs of TGNN are a sequence of feature vectors $\boldsymbol{X}_{t_s,t_e} := [X^{(t_s)}, \cdots, X^{(t_e)}]$ and an adjacency matrix $A \in \mathcal{A} := \{0,1\}^{|V|\times|V|}$. Here, t_s, t_e and $X^{(t)} \in \mathbb{R}^{|V|\times F}$ denote the starting time, the ending time, and the input feature sequence. The model is referred to by its forwarding function $\Phi : \mathcal{X} \times \mathcal{A} \to \mathcal{Y}$, where $\mathcal{X}$ and $\mathcal{Y}$ are the spaces of the input feature sequence and the output.

We use the general formulation of GNNs based on the message passing mechanism [55], which involves 3 computations: propagation, aggregation, and update:

$$\begin{aligned} m_{ij}^{(l)} &= \mathrm{MSG}\left(h_i^{(l-1)}, h_j^{(l-1)}\right), \\ a_i^{(l)} &= \mathrm{AGG}\left(\left\{m_{ji}^{(l)}\right\}_{j\in\mathcal{N}_i}\right) \\ h_i^{(l)} &= \mathrm{UPD}\left(a_i^{(l)}, h_i^{(l-1)}\right) \end{aligned}$$

where m_{ij} is the message from node i to node j, $h_i^{(l)}$ is the hidden representations of node i at layer l and $\mathcal{N}_i$ is node i's neighbors. The propagation computes the message from node i to node j if there is an edge between them: $m_{ij}^{(l)} = \mathrm{MSG}_l(h_i^{(l-1)}, h_j^{(l-1)})$, where $h_i^{(l-1)}$ and $h_j^{(l-1)}$ is the hidden representations of node i and j at layer $l-1$. The aggregation calculates the aggregated message at node i: $a_i^{(l)} = \mathrm{AGG}_l(m_{ji}^{(l)} \mid j \in \mathcal{N}_i)$, where $\mathcal{N}_i$ is the neighbors of node i. The update transforms both the aggregated message and the previous hidden representations into the current representations $h_i^{(l)} = \text{UPD}_l(a_i^{(l)}, h_i^{(l-1)})$. The final representation at the last layer L, $h_i^{(L)}$, is commonly used to generate a prediction, i.e. $Y = \text{READOUT}(h_i^{(L)})$. Typically, the MSG, UPD, and READOUT functions consist of trainable weights and biases followed by an activation function. Some common choices for the AGG are max, mean, and concatenating operations.

By denoting $\bar{\Phi}$ to be the forwarding function of a GNN and $H^{(t)}$ to be the temporal messages, we can formulate the forwarding function $\Phi : \mathcal{X} \times \mathcal{A} \to \mathcal{Y}$ of a TGNN based on its sequential implementation [162]:

$$\begin{aligned} H^{(t_s)} &= \bar{\Phi}(X^{(t_s)}, A) \\ H^{(t)} &= \bar{\Phi}(H^{(t-1)}, X^{(t)}, A),\ t = t_s + 1, \ldots t_e \end{aligned} \tag{6-1}$$

The *base* GNN $\bar{\Phi}$ typically consists of some graph layers followed by a readout. The output Y is computed either by applying a readout on the temporal message at the last layer $H^{(t_e)}$ or from the node's final hidden features. In this manuscript, capital letters, e.g. X, Y, and H, refer to external signals of the GNN blocks, while small letters, e.g. m, a and h, are for internal signals.

This work studies black-box explainers of GNNs and TGNNs based on the type of perturbations that the explainers use:

1. Node-perturbation class $\mathcal{G}_v$: the explainer can perturb the entries of the feature matrices in $\boldsymbol{X}_{t_s,t_e}$.
2. Edge-perturbation class $\mathcal{G}_e$: the explainer can remove some edges from the input adjacency matrix A.
3. Node-and-Edge-perturbation class $\mathcal{G}_a$: the explainer can perturb both the feature matrices in $\boldsymbol{X}_{t_s,t_e}$ and remove some edges in the input adjacency matrix A.

The usage of Dynamic Bayesian Networks (DBNs) in this work is to model internal computations of TGNNs so that theoretical analysis can be conducted. A DBN [26] can be considered as an extension of Bayesian networks (BNs) [102] to model the temporal dependency of systems' variables. Temporal information is integrated via edges between adjacent time steps.

Figs. 6-2a shows an example of a DBN: the Two-Timeslice Bayesian Network (2TBN) [97]. Its equivalent BN in the form of unrolled 4-time-step BN is shown in Figs. 6-2b. We provide a brief introduction of BNs and DBNs in Appendix G. Readers can find more rigorous descriptions of DBNs in [97].

6.3 Unidentifiable Proof for Neural Networks

Given a black-box model Φ and a class of explanation methods, the Unidentifiable Proof formalizes the idea that certain information of Φ cannot be identified and used as the explanation by a class of explainers. Before describing the Unidentifiable Proof, we need to formalize the *ground-truth explanation*. The first two subsections discussing about the *interpretable domain* and the *Transparent Model* serve that purpose. Intuitively, the interpretable domain is the domain of all available explanations and the Transparent Model is a domain's member that can faithfully capture the model. The latter part of this section describes the Unidentifiable Proof. The general idea of the Unidentifiable Proof is by construction: it constructs two instances of the model whose Transparent Models are different; however, the information extracted from them by a given class of explainers is exactly the same. This means no explainer of that class can identify the information differentiating the two Transparent Models. This gives us formal notions of unidentifiable information.

6.3.1 The Interpretable Domain

Given a black-box model Φ and an input $\boldsymbol{X}$, the explainers solve for an interpretable representation of the prediction $\Phi(\boldsymbol{X})$, denoted as $g(\Phi(\boldsymbol{X}))$. For the sake of explaining, $g(\Phi(\boldsymbol{X}))$ should be intuitive and interpretable; therefore, we call the space of $g(\Phi(\boldsymbol{X}))$ the *interpretable domain*. For example, the interpretable domains for GNNs have been chosen to be a set of scores on some nodes/edges' features, the set of linear functions, and the set of probabilistic models on the input's nodes [33, 140, 61]. Intuitively, a good interpretable domain should balance its representative power and interpretability. In this work, we consider it to be the set of DBNs. We describe how DBNs can help explain TGNNs in the next subsection.

6.3.2 The Transparent Model

Given an interpretable domain and a black-box model, there is no guarantee that there exists an interpretable representation that correctly explains Φ. Nevertheless, in some specific contexts, an interpretable representation that can fully describe the black-box model exists. Particularly, the work [126] embeds a linear function inside a black-box model, which means that a linear function can faithfully describe and explain that black-box. This implies, for a given interpretable domain and for some Φ, an explanation that can fully explain Φ exists. We denote it by *the Transparent Model* $\mathcal{I}$. In some cases, the Transparent Model only exists for a subset of inputs $\mathcal{S} \subseteq \mathcal{X}$. We write the Transparent Model in those cases as $\mathcal{I}(\Phi(\boldsymbol{X})), \boldsymbol{X} \in \mathcal{S}$. Furthermore, we call the assumption $\mathcal{I}(\Phi(\boldsymbol{X}))$ exists the *Existence assumption*.

We now discuss the Transparent Model $\mathcal{I}(\Phi)$ in terms of DBNs. For all our Unidentifiable Proofs, the target of explanation will be the prediction on a node of the input graph. The explanation will be in the form of a DBN $\mathcal{B}$, whose variables are associated with the corresponding nodes in the input graph. As each node of the input graph is physically associated with a distinct set of neurons in the graph layers of the TGNN, we associate each variable of $\mathcal{B}$ to the sending messages of the neurons corresponding to that node in the TGNN. Note that the sending messages from a node consist of not only internal messages in the graph layers but also temporal messages and output messages. These associations allow us to capture the dynamics of the TGNN via DBN. Fig. 6-3 provides an illustration of these associations.

We say a DBN $\mathcal{B}$ is the Transparent Model of a model Φ if (i) all independence claims of $\mathcal{B}$ about its variables are consistent with the messages sent from the corresponding neurons in the model and (ii) $\mathcal{B}$ is minimal. Condition (i) is obvious since incorrect claims from the explanation are undesirable. Note that, this condition implies the DBN $\mathcal{B}$ can represent all communicating messages during the forward computation of Φ. Condition (ii) enforces the explaining DBN to be as informative as possible, i.e. it should remove unnecessary edges when they do not help explain the model's computations.

6.3.3 The Unidentifiable Proof

Under the Existence assumption, i.e. $\mathcal{I}(\Phi(\boldsymbol{X}))$ exists, a good explanation method g is expected to return $g(\Phi(\boldsymbol{X}))$ to be similar to $\mathcal{I}(\Phi(\boldsymbol{X}))$. This provides us a necessary condition to theoretically analyze the limits of explanation methods: given two models Φ_1 and Φ_2 with distinctively different transparent models $\mathcal{I}(\Phi_1)$ and $\mathcal{I}(\Phi_2)$, a good explainer must return different explanations, i.e. $g(\Phi_1(\boldsymbol{X})) \neq g(\Phi_2(\boldsymbol{X}))$. This necessary condition is illustrated via Fig. 6-4.

The above necessary conditions can be formalized as follows. Given two neural networks Φ_1 and Φ_2, the Unidentifiable Proof holds if their Transparent Models exist and:

$$g(\Phi_1(\boldsymbol{X})) = g(\Phi_2(\boldsymbol{X})),\ \forall g \in \mathcal{G}, \forall \boldsymbol{X} \in \mathcal{S} \subseteq \mathcal{X} \tag{6-2}$$

$$\exists \boldsymbol{X} \in \mathcal{S} \subseteq \mathcal{X} \text{ s.t } \mathcal{I}(\Phi_1(\boldsymbol{X})) \neq \mathcal{I}(\Phi_2(\boldsymbol{X})) \tag{6-3}$$

The first condition says the explanations of the two models provided by all explainers in $\mathcal{G}$ are the same. The condition can be shown by examining the forwarding computations of the two models. The second condition states the existence of some inputs such that their Transparent Models are different. The main challenge in proving that condition is in concretely determining $\mathcal{I}(\Phi_1(\boldsymbol{X}))$ and $\mathcal{I}(\Phi_2(\boldsymbol{X}))$. The two conditions then imply the explainer cannot learn the Transparent Model of at least one of the two models. More importantly, as g outputs the same information in explaining both models, any information that can be used to differentiate the two models cannot be inferred from g. Thus, all information differentiating $\mathcal{I}(\Phi_1(\boldsymbol{X}))$ and $\mathcal{I}(\Phi_2(\boldsymbol{X}))$ cannot be inferred from the explainer either. The arguments, therefore, establish the unidentifiable information for the class of explainers $\mathcal{G}$.

6.4 Unidentifiable Proof for Node-Perturbation

We now provide the Unidentifiable Proof for the Node-perturbation class $\mathcal{G}_v$. We show that for a simple max computation conducted by the TGNNs, Node-perturbation cannot identify the messages' propagating paths carrying out the predictions in the model. We also elaborate on how the result can be applied to GNNs in Sect. 6.8.

6.4.1 The Training Task

In this construction, the TGNNs operate on a graph of 4 nodes forming a square, with the following adjacency matrix:

$$A = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

The training task is to recognize the maximum positive inputs of node 3 and return the result at node 1:

$$\begin{aligned} Y_1^{(t)} &= \max\{0 \text{ and } X_3^{(t')}, 0 \leq t' \leq t\} \\ Y_2^{(t)} &= Y_3^{(t)} = Y_4^{(t)} = 0 \end{aligned} \tag{6-4}$$

The model's input and output at each time-step t are both in $\mathbb{R}^4$. Fig. 6-1a provides an illustration of this training task.

6.4.2 The Models

We construct two TGNNs, Φ_1^v and Φ_2^v, with the same architecture but different parameters. We consider the input's length $T = 2$ for the sake of brevity. Each node i is associated with a hidden feature vector $\boldsymbol{h}_i = [hr_i, ht_i, hs_i, hz_i, ho_i^+, ho_i^-] \in \mathbb{R}^6$, whose features mean:

1. hr_i: message that node i receives.

2. ht_i: temporal message that node i receives.

3. hs_i: feature determining if node i sends message.

4. hz_i: feature determining if node i outputs zero.

5. ho_i^+ and ho_i^-: features determining the output of node i.

During all computations, h_s and h_z are constant. In practice, they can be the results of a zero weight combined with a constant bias. Their values in the two constructed models are shown in

Table 6-2. The upcoming construction will ensure that, if $hs_i = k_s$, node i does not send any message, and if $hz_i = k_z$, the output of node i will be zero.

Our proposed TGNN architecture has 2 graph layers followed by a READOUT layer (Fig. 6-5). By conventions, we use $l \in \{0,1,2\}$ to indicates the model's graph layers with $h_i^{(t,l=0)}$ refers to the model's input:

$$\boldsymbol{h}_i^{(t,l=0)} = \left[X_i^{(t)}, H_i^{(t-1)}, *, *, 0, 0\right] \tag{6-5}$$

where $*$ means the features are determined by the model's weights and biases, i.e. by hs and hz. The temporal signal $H^{(t-1)}$ has the same dimension as the model's output Y.

The two models have trainable weights and biases such that the MSG and AGG functions work as follows:

$$m_{ji}^{(t,l)} = \text{ReLU}\left(hr_j^{(t,l-1)} - hs_j^{(t,l-1)}\right) \tag{6-6}$$

$$a_i^{(t,l)} = \sum_{j \in \mathcal{N}_i} m_{ji}^{(t,l)} \tag{6-7}$$

for $l \in \{1,2\}$. This means, if $hs_j = k_s$ is large and unchanged, there is no message coming out of node j. Thus, the a_i consists of messages only from node j with $hs_j = 0$.

Meanwhile, the UPD returns $\boldsymbol{h}_i^{(t,l)}$, which is $\text{ReLU}\left(\boldsymbol{w}_h^\top \boldsymbol{h}_i^{(t,l-1)} + w_a a_i^{(t,l)}\right)$. The parameters are chosen such that:

$$hr_i^{(t,l)} = \text{ReLU}\left(a_i^{(t,l)}\right), ht_i^{(t,l)} = \text{ReLU}\left(ht_i^{(t,l-1)}\right) \tag{6-8}$$

$$hs_i^{(t,l)} = \text{ReLU}\left(hs_i^{(t,l-1)}\right), hz_i^{(t,l)} = \text{ReLU}\left(hz_i^{(t,l-1)}\right)$$

$$ho_i^{\pm(t,l)} = \text{ReLU}\left(\pm a_i^{(t,l)} \mp ht_i^{(t,l-1)}\right) \tag{6-9}$$

$$H_i^{(t)} = \text{ReLU}\left((hr_i + ht_i + ho_i^+ + ho_i^-)/2 - hz_i\right) \tag{6-10}$$

Finally, the prediction of the model Y is assigned to $H^{(t=2)}$. We can check that h_t, h_s, and h_z are unchanged during the forwarding computation and hr_i is indeed the received signal at node i with the above specification.

6.4.3 The Forwarding Computation of Φ_1^v and Φ_2^v.

To show that the forwarding computations of the two constructed models satisfy the training tasks, we first pay attention to ho_i^+ and ho_i^-, whose summation is the difference between $a_i^{(t,l)}$ and $ht_i^{(t,l-1)}$. Thus, the maximum signal that node i received is:

$$\begin{aligned}
&\frac{1}{2}\left(hr_i^{(t,L)}+ht_i^{(t,L)}+ho_i^{+(t,L)}+ho_i^{-(t,L)}\right)\\
=&\frac{1}{2}\left(a_i^{(t,L)}+H_i^{(t-1)}+\left|a_i^{(t,l)}-H_i^{(t-1)}\right|\right)\\
=&\max\left\{a_i^{(t,L)},H_i^{(t-1)}\right\}=\max\left\{hr_i^{(t,L)},H_i^{(t-1)}\right\}
\end{aligned}\tag{6-11}$$

Due to the READOUT in (6-10), the outputs of nodes with large hz_i are zero. Thus, from Table 6-2, we have:

$$H_i^{(t)}=\begin{cases}\max\left\{hr_i^{(t,L)},H_i^{(t-1)}\right\} & \text{for } i=1\\ 0 & \text{for } i\in\{2,3,4\}\end{cases}\tag{6-12}$$

As the prediction of the model Y is set to $H^{(t=2)}$, to show that the models work as specified in the training tasks, we need to verify that $hr_1^{(t,L)}=X_3^{(t)}$ for all t. We now show the claim for Φ_1^v.

For $l=1$, $m_{32}^{(t,1)}=hr_3^{(t,0)}=X_3^{(t)}$ from (6-6). Thus, $hr_2^{(t,1)}=a_2^{(t,1)}=m_{32}^{(t,1)}=X_3^{(t)}$ ((6-7) and (6-8)). For $l=2$, $m_{21}^{(t,2)}=hr_2^{(t,1)}=X_3^{(t)}$ and $hr_1^{(t,2)}=a_1^{(t,2)}=m_{21}^{(t,2)}=X_3^{(t)}$. Therefore, Φ_1^v fulfills the training task. Note that $a_2^{(t,1)}=m_{32}^{(t,1)}$ and $a_1^{(t,2)}=m_{21}^{(t,2)}$ because there is no message sending from node 1 and node 4. The claim for Φ_2^v trivially follows by swapping node 2 with node 4.

6.4.4 The Transparent Models of Φ_1^v and Φ_2^v

We now examine the Transparent models $\mathcal{I}(\Phi_1^v(\boldsymbol{X}))$ and $\mathcal{I}(\Phi_2^v(\boldsymbol{X}))$. Fig. 6-6 shows two DBNs representing the messages coming out of the model's nodes. Particularly, the variable of node i at time t, denoted by $\mathcal{V}_i^t$, represents $m_{ij}^{(t,l)}$ $(\forall l,\forall j\in\mathcal{N}_i)$ and the $H_i^{(t)}$. Our claim is the two DBNs can faithfully explain the two models when their inputs are bounded by $K:=\min\{k_s,k_z\}$:

Lemma 6-1. *The DBN $\mathcal{B}_1^v$ ($\mathcal{B}_2^v$) in Fig. 6-6 can embed all information of the hidden features of TGNN Φ_1^v (Φ_2^v) without any loss when the input signal is bounded by $K:=\min\{k_s,k_z\}$. Furthermore, the DBN is minimal.*

Proof. To show that the DBN $\mathcal{B}_1^v$ can represent Φ_1^v without any loss, we show:

1. $\mathcal{B}_1^v$ can express how the predictions $H_i^{(t)}$ are generated.

2. $\mathcal{B}_1^v$ can express how the messages propagate in the model.

The first claim only involves node $i = 1$ (as $H_i^{(t)} = 0$ for all other nodes). Because $H_1^{(t)} = \max\left\{X_3^{(t)}, H_1^{(t-1)}\right\}$ (shown below (6-12)), the paths from $\mathcal{V}_3^t$ to $\mathcal{V}_1^t$ and from $\mathcal{V}_1^{t-1}$ to $\mathcal{V}_1^t$ are sufficient to represent how the predictions are generated.

From Table 6-2, we observe that, as long as the inputs are bounded by K, only nodes 2 and 3 send messages. Thus, to show the second claim, we only need to consider messages from those nodes. It is clear from (6-6), (6-7) and (6-8) that:

$$\begin{aligned}
&X_2^{(t)} \to m_{21}^{(t,1)} \to a_1^{(t,1)} \to hr_1^{(t,1)} \to \varnothing \\
&X_2^{(t)} \to m_{23}^{(t,1)} \to a_3^{(t,1)} \to hr_3^{(t,1)} \to m_{32}^{(t,2)} \text{ and } m_{34}^{(t,2)} \\
&X_3^{(t)} \to m_{32}^{(t,1)} \to a_2^{(t,1)} \to hr_2^{(t,1)} \to m_{23}^{(t,2)} \text{ and } m_{21}^{(t,2)} \\
&X_3^{(t)} \to m_{34}^{(t,1)} \to a_4^{(t,1)} \to hr_4^{(t,1)} \to \varnothing
\end{aligned}$$

where the arrow means *determining* and $\to \varnothing$ means the signals result in no other messages. We can see that the messages sent out from nodes 2 and 3 are only dependent on the signals of those nodes. As those dependencies can be captured by an edge between $\mathcal{V}_2^t$ and $\mathcal{V}_3^t$, we have the claim.

The above arguments also show that $\mathcal{B}_1^v$ is minimal. Specifically, the edges $(\mathcal{V}_1^{t-1}, \mathcal{V}_1^t)$ and $(\mathcal{V}_2^t, \mathcal{V}_3^t)$ are necessary because of the temporal dependency $H_1^{(t-1)} \to H_1^{(t)}$ and the messages' dependency between nodes 2 and 3. We then require a path from $\mathcal{V}_3^t$ to $\mathcal{V}_1^t$ to capture the dependency $X_3^{(t)} \to H_1^{(t)}$ when $X_3^{(t)} > H_1^{(t-1)}$. Thus, at least another edge is needed. Since $\mathcal{B}_1^v$ has 3 edges, it is minimal. $\square$

From Lemma 6-1, we write $\mathcal{B}_1^v = \mathcal{I}(\Phi_1^v(\boldsymbol{X}))$ and $\mathcal{B}_2^v = \mathcal{I}(\Phi_2^v(\boldsymbol{X}))$ for all $\boldsymbol{X}$ whose entries are bounded by K.

6.4.5 Unidentifiable Proof

As the two DBNs contain distinct information regarding $\mathcal{V}_2$ and $\mathcal{V}_4$, for an explainer that is capable of explaining the two corresponding models Φ_1^v and Φ_2^v, it must be able to differentiate the two DBNs. Unfortunately, in the next Lemma 6-2, we show that the outputs of the two TGNNs are the same under node-perturbation; hence, explainers of the class cannot explain them:

Lemma 6-2. *For all $\boldsymbol{X}$ such that $X_i^{(t)} \leq \min\{k_s, k_z\}$, we have $\Phi_1^v(\boldsymbol{X}) = \Phi_2^v(\boldsymbol{X})$.*

Proof. From the examination of the forwarding computations, we know that both models satisfy (6-4) for all $\boldsymbol{X}$ bounded by $\min\{k_s, k_z\}$. Thus, their outputs on such $\boldsymbol{X}$ are the same. Thus, we have the Lemma. □

We are now ready for the unidentifiable result of Node-perturbation:

Theorem 6-1. *For a TGNN Φ, denote $\mathcal{P} := \{(X, A, \Phi(X,A)) | X_i \leq K\}_X$, i.e. the set of Node-perturbation-response of Φ when the perturbations are bounded by K. Denote g an algorithm accepting $\mathcal{P}$ as inputs. For any $K > 0$ and g, there exists a Φ such that:*

1. *For the interpretable domain of DBNs, the Transparent Model of Φ exists for all inputs in $\mathcal{P}$.*

2. *g cannot determine the Transparent Model of Φ.*

Proof. We first set k_s and k_z (Table 6-2) to K. We then construct Φ_1^v and Φ_2^v as described from (6-5) to (6-10). Denote $\mathcal{P}_1$ and $\mathcal{P}_2$ the sets of Node perturbation-response of Φ_1^v and Φ_2^v, respectively. Note that, from Lemma 6-1, we have $\mathcal{B}_1^v$ and $\mathcal{B}_2^v$ are the Transparent Models of Φ_1^v and Φ_2^v.

Given a Node-perturbation-response, suppose g returns either $\mathcal{B}_1^v$ or $\mathcal{B}_2^v$ (or equivalents claims on which DBN is fitter). As $\mathcal{P}_1$ is the same as $\mathcal{P}_2$ (Lemma 6-2), the outputs of g on the 2 perturbation-response sets must be the same. If, for example, $g(\mathcal{P}_1) = \mathcal{B}_1^v$, g cannot determine that $\mathcal{B}_2^v$ is the Transparent Model for Φ_2^v as $g(\mathcal{P}_2) = g(\mathcal{P}_1) = \mathcal{B}_1^v$. Thus, selecting Φ_2^v as Φ proves the Theorem. □

From the proof of Theorem 6-1, we see that, even though Φ_1^v and Φ_2^v operate on different paths (reflected in the difference between $\mathcal{B}_1^v$ and $\mathcal{B}_2^v$), all explanations produced by methods in $\mathcal{G}_v$ cannot differentiate Φ_1^v and Φ_2^v. Therefore, we can conclude that Node-perturbation methods are not able to reliably identify which paths carry out the model's predictions.

6.5 Unidentifiable Proof for Edge-Perturbation

This section is about the Unidentifiable Proof for Edge-perturbation class $\mathcal{G}_e$. We show that removing edges from input graphs is not enough to identify all nodes contributing to a max operation conducted by the TGNNs. The intuition is, if the messages are gated by the features, edge perturbation does not reveal the sources of those messages.

6.5.1 The Training Task and the Models

Our proof considers a graph of 3 nodes forming a line. The task (Fig. 6-1b) is to recognize the maximum positive inputs observed in nodes 2 and 3, and return results at node 1:

$$Y_1^{(t)} = \max\left\{0, X_2^{(t')} \text{ and } X_3^{(t')}, 0 \le t' \le t\right\} \tag{6-13}$$

The outputs on other nodes are zeros.

We use the same architecture as in Sect. 6.4 (Fig. 6-5) to construct two TGNNs named Φ_1^e and Φ_2^e. The hidden vector of each node has 5 main features, i.e. $\boldsymbol{h}_i = [hr_i, ht_i, hs_i, hz_i, hl_i]$, and 6 additional features just for output purposes, denoted by $\boldsymbol{ha}_i = [hrl_i^+, hrl_i^-, hrt_i^+, hrt_i^-, hlt_i^+, hlt_i^-]$. The only new feature in $\boldsymbol{h}_i$ compared to the previous construction in Node-perturbation is hl_i, which is the lag version of hr_i: $hl_i^{(t,l)} = \text{ ReLU}\left(hr_i^{(t,l-1)}\right)$. Φ_1^e and Φ_2^e use the same MSG, AGG, and UPD functions as described from (6-6) to (6-9). The difference between Φ_1^e and Φ_2^e is only in node 3: while it sends a message in Φ_1^e (as $hs_3 = 0$), it does not in Φ_2^e (as $hs_3 = k_s$).

Regarding the 6 additional features $\boldsymbol{ha}_i$, they are zeros at initialization. Their updates are:

$$\begin{aligned}
hrl_i^{\pm(t,l)} =& \text{ReLU}\left(\pm\left(a_i^{(t,l)} - hr_i^{(t,l-1)}\right)\right)\\
hrt_i^{\pm(t,l)} =& \text{ReLU}\left(\pm\left(a_i^{(t,l)} - ht_i^{(t,l-1)}\right)\right)\\
hlt_i^{\pm(t,l)} =& \text{ReLU}\left(\pm\left(hr_i^{(t,l-1)} - ht_i^{(t,l-1)}\right)\right)
\end{aligned}$$

Additionally, the READOUT and the prediction are:

$$H_i^{(t)} = \text{ReLU}\left(\tau_i^{(t,l=2)} - hz_i^{(t,l=2)}\right), \quad Y = H^{(t=2)} \tag{6-14}$$

where $\tau_i^{(t,l)} := \frac{1}{3}(\mathbf{1}^\top \boldsymbol{ha}_i^{(t,l)} + hr_i^{(t,l)} + hl_i^{(t,l)} + ht_i^{(t,l)})$. The goal of the above setting is to make

$$H_1^{(t)} = \max\left\{X_3^{(t)}, X_2^{(t)}, H_1^{(t-1)}\right\}$$

and $H_i^{(t)} = 0$ for $i \in \{2,3\}$, i.e., to make ϕ_1^e satisfy the training task (6-13) as $Y = H^{(t=2)}$.

Since $hs_3 = k_s$ in Φ_2^e, node 3 does not send messages. This makes $hr_1^{(t,l=2)} = 0$ as there is no message coming to node 1 at $l = 2$. This makes the output of Φ_2^e at node 1 equal to:

$$H_1^{(t)} = \max\left\{hl_1^{(t,L)}, H_1^{(t-1)}\right\} = \max\left\{X_2^{(t)}, H_1^{(t-1)}\right\}$$

Therefore, by assigning the output Y to $H^{(t)}$, we make the output of Φ_2^e on node 1 equal:

$$Y_1^{(t)} = \max\left\{0 \text{ and } X_2^{(t')}, 0 \leq t' \leq t\right\} \tag{6-15}$$

By comparing (6-13) to (6-15), it is clear that Φ_1^e and Φ_2^e are different. However, when $X_2^{(t)} > X_3^{(t)}$, the responses of Φ_1^e and Φ_2^e are the same even when some edges are removed from the input graph. We state that observation below:

Lemma 6-3. *For the task in Fig 6-1b, denote $\bar{A}$ the adjacency matrix obtained by either keeping the input adjacency matrix A unchanged or by removing some edges. For any given $\boldsymbol{X}$ such that $X_i^{(t)} \leq \min\{k_s, k_z\}$ and $X_2^{(t)} > X_3^{(t)}$, we have $\Phi_1^e(\boldsymbol{X}, \bar{A}) = \Phi_2^e(\boldsymbol{X}, \bar{A})$.*

Proof. We only need to consider the output at node 1 since the outputs of all other nodes are 0 (as $hz_i = k_z$ for $i \in \{2,3\}$. If no edge is removed, from the analysis of the forwarding computation (below (6-14)), we know that both models return the maximum of $X_2^{(t)}$ at node 1 as $X_2^{(t)} > X_3^{(t)}$. If the edge between node 1 and node 2 is removed, there is no message coming to node 1 and $hr_1^{(t,l=2)}$ in both models will be 0. The remaining case is when only the edge between nodes 2 and 3 is removed. In this case, Φ_1^e simply becomes Φ_2^e and their outputs must be the same. We then have the Lemma. □

6.5.2 The Transparent Models and Unidentifiable Proof

As Φ_1^e is different with Φ_1^v only in node 4 and the additional content in the propagating messages, it follows that $\mathcal{B}_1^e$ (Fig. 6-7) is the Transparent Model of Φ_1^e for $\boldsymbol{X}$ bounded by $\min\{k_s, k_z\}$. We write $\mathcal{B}_1^e = \mathcal{I}(\Phi_1^e)$. Note that even when $X_2^{(t)} > X_3^{(t)}$, $m_{21}^{(t,l=1)}$ is determined by $X_3^{(t)}$. Thus, the edge between $\mathcal{V}_3^t$ and $\mathcal{V}_2^t$ in $\mathcal{B}_1^e$ is necessary.

Regarding Φ_2^e, as it is just Φ_1^e with node 3 disconnected, $\mathcal{B}_2^e$ (Fig. 6-7) is the Transparent Model of Φ_2^e, i.e. $\mathcal{B}_2^e = \mathcal{I}(\Phi_2^e)$. The above arguments combined with Lemma 6-3 give us the following Theorem about the Unidentifiable Proof for Edge-perturbation:

Theorem 6-2. *For a TGNN Φ, denote $\mathcal{P} := \{(X, \bar{A}, \Phi(X, \bar{A}))|X_i \le K\}_{\bar{A}}$, i.e. the set of Edge-perturbation-response of Φ where $\boldsymbol{X}$ are fixed, and $\bar{A}$ is defined as in Lemma 6-3. Denote g an algorithm accepting $\mathcal{P}$ as inputs. For any $K > 0$ and g, there exists a Φ satisfying the two conditions in Theorem 6-1.*

Proof. We first choose k_s and k_z in the node's features to K in the Theorem. We then construct Φ_1^e and Φ_2^e as described above. Denote $\mathcal{P}_1$ and $\mathcal{P}_2$ the sets of Edge-perturbation-response of Φ_1^e and Φ_2^e, respectively. From the discussion of Transparent Models, we have $\mathcal{B}_1^e$ and $\mathcal{B}_2^e$ are the Transparent Models of the two TGNNs.

Given an Edge-perturbation-response, suppose g returns either $\mathcal{B}_1^e$ or $\mathcal{B}_2^e$. Due to Lemma 6-3, the Edge-perturbation-response $\mathcal{P}_1$ is the same as $\mathcal{P}_2$; therefore, the outputs of g on the 2 perturbation-response sets must be the same. Hence, similar arguments as in the proof of Theorem 6-1 give us Theorem 6-2. □

From the proof of Theorem 6-2, we can see that when the propagating path of TGNN is gated by an intermediate node, the path cannot be identified by $\mathcal{G}_e$. This means Edge-perturbation might not be faithful in detecting all graph features contributing to TGNN's predictions.

6.6 Unidentifiable Proof for Node-and-Edge Perturbation in TGNN

This section provides the Unidentifiable Proof for the Node-and-Edge-perturbation. The proof shows perturbing both nodes and edges is not sufficient to identify which nodes carry out the temporal aggregation in TGNNs.

6.6.1 The Training Task

The TGNNs operate on a line graph (Fig. 6-1c). The task is to record the maximum positive inputs observed in node 3 and return the result at node 1:

$$Y_1^{(t)} = \max\{0 \text{ and } X_3^{(t')}, 0 \le t' \le t\} \tag{6-16}$$

The outputs on other nodes are zeros. This proof constructs 2 TGNNs whose internal behaviors are described by the DBNs shown in Fig. 6-8. The main difference between this proof compared to the previous is that the models involve temporal messages.

6.6.2 The Models

We use the same architecture as in Fig. 6-5. The hidden feature vectors have 7 features, i.e. $\boldsymbol{h}_i = [hr_i, ht_i, hs_i, hz_i, ho, ho_i^+, ho_i^-]$. Except for the newly introduced ho_i, all features have the same meaning as described in Sect. 6.4. The two constructed models, called Φ_1^a and Φ_2^a, have different MSG functions, READOUT functions, and hidden constant features, i.e. hs_i and ho_i. The constant features for the two models are shown in Table 6-3.

The MSG, AGG, UPD, and READOUT of Φ_1^a are as specified from (6-6) to (6-10). Since ho_i in Φ_1^a is just a dummy variable, Φ_1^a satisfies (6-16) since Φ_1^a is Φ_1^v without node 4.

In Φ_2^a, we use hz_i to control the temporal messages $H^{(t)}$ and the newly introduced ho_i is to control the output Y. We will show that the assignment in Table 6-3 will make node 3 to be the one that conducts the temporal aggregation in Φ_2^a. The model uses the same AGG and UPD functions as specified from (6-7) to (6-9). The update rule of ho_i is $ho_i^{(t,l)} = \text{ReLU}(ho_i^{(t,l-1)})$. The MSG function has the trainable weight $\boldsymbol{w}_m$ such that:

$$m_{ij}^{(t,l)} = \text{ReLU}\left(hr_i + ht_i + ho_i^+ + ho_i^-)/2 - hs_i\right) \tag{6-17}$$

Here, all variables on the RHS have temporal index t and layer index l. The final difference in Φ_2 compared to Φ_1 is its READOUT as we set $H_i^{(t)} = \text{ReLU}\left(hr_i - hz_i\right)$ and $Y_i = \text{ReLU}\left(hr_i - ho_i\right)$.

6.6.3 Forwarding computation of Φ_2^a

We now show that the forwarding computation of Φ_2^a fulfills (6-16). We start with the following observations:

1. The message m_{ij} (6-17) is the maximum of hr_i and ht_i (similar to arguments at (6-11)) when $hs_i = 0$.

2. Because of the READOUT (specified below (6-17)), $H_2^{(2)}$ and ht_2 are always zeros. Thus, m_{2j} is always hr_2.

3. Since node 2 only receives messages from node 3 (as $hs_1 = 0$) and the input message, $hr_2^{(t,l)}$ is always the message sent from 3 for l odd. Combining with the above point, $m_{21}^{(t,l=2)}$ is always $m_{32}^{(t,l=1)}$.

4. As node 1 only connects to node 2, $hr_1^{(t,l=2)}$ is always $m_{21}^{(t,l=2)}$, which is $m_{32}^{(t,l=1)}$.

From those observations, we have $hr_1^{(t,l=2)}$ is the maximum of $hr_3^{(t,l=0)}$ and ht_3. If ht_3 is always the maximum input in the past and $hr_3^{(t,l=0)}$ is the current input of node 3, then we have the model fulfill the training task (Fig. 6-1c).

To see that ht_3 is always the maximum input in the past, we refer to Table 6-4 tracking the received message hr_i and the out-going message m_{i*} for arbitrary inputs $X^{(t=1)} = [\alpha_1, \alpha_2, \alpha_3]$ and $X^{(t=2)} = [\beta_1, \beta_2, \beta_3]$. We first consider the claim for the input's length $T = 2$. For the first time-step $t = 1$, the sending and receiving messages can be deduced in the same manner as in Φ_1^v (Sect. 6.4). For $t = 2$, we have $ht_3 = \alpha_3$ (because only $hz_3 = 0$), which is indeed the maximum signal in the past of node 3. For larger T, we can further examine the Table 6-4 and deduce that claim: at $l = 0$, node 3 sends out $\max\{X_3^{(t)}, H_3^{(t-1)}\}$. At $l = 1$, this message is received at node 2. Finally, this message is sent back to node 3 at $l = 2$.

6.6.4 The Transparent Models of Φ_1^a and Φ_2^a

As Φ_1^a is Φ_1^v without node 4, we have the $\mathcal{B}_1^a$, which is $\mathcal{B}_1^v$ without variables for node 4, is the Transparent Model of Φ_1^a. We write $\mathcal{B}_1^a = \mathcal{I}(\Phi_1^a(\boldsymbol{X}))$ for $\boldsymbol{X}$ bounded by $\min\{k_s, k_z\}$.

On the other hand, the transparent model of Φ_2^a can be shown to be the DBN $\mathcal{B}_2^a$ in Fig. 6-8 by the following Lemma:

Lemma 6-4. *The DBN $\mathcal{B}_2^a$ in Fig. 6-8 can embed all information of the hidden features of TGNN Φ_2^a without any loss when the input signal is bounded by $K := \min\{k_s, k_z\}$. Furthermore, the DBN is minimal.*

Proof. The proof has the same structure as in Lemma 6-1, in which we show the DBN can express the predictions and the messages. For the prediction, in the forwarding computation, we have shown $hr_1^{(t,l=2)}$ is the maximum of $hr_3^{(t,l=0)}$ and ht_3. Thus, the path $\mathcal{V}_3^{t-1} - \mathcal{V}_3^t - \mathcal{V}_2^t - \mathcal{V}_1^t$ is sufficient to express the prediction. For the messages, since only nodes 2 and 3 are sending out messages, we only need the edge $\mathcal{V}_3^t - \mathcal{V}_2^t$ and edge $\mathcal{V}_2^t - \mathcal{V}_1^t$ to represent them. We can also track the signal via Table 6-4 to verify this.

It is straightforward to verify that the DBN $\mathcal{B}_2^a$ is a minimal solution: simply from the fact that the prediction at node 1 depends on the past signal ht_3, which means a path between $\mathcal{V}_3^{t-1}$ and $\mathcal{V}_1^t$ must be maintained. Thus, we cannot remove any edges from $\mathcal{B}_2^a$ while keeping it consistent with Φ_2^a. □

As $\mathcal{B}_1^a$ and $\mathcal{B}_2^a$ contain different information, e.g. different set of independent variables, Lemma 6-4 allows us to claim $\mathcal{I}(\Phi_1^a(\boldsymbol{X})) \neq \mathcal{I}(\Phi_2^a(\boldsymbol{X}))$ for some $\boldsymbol{X}$ bounded by K.

6.6.5 Unidentifiable Proof

Similar to previous proofs, we show that, for all $\boldsymbol{X}$ bounded by K and a valid adjacency matrix, the outputs of the two constructed models are equal in the following Lemma:

Lemma 6-5. *For the training task in Fig 6-1c, denote $\bar{A}$ the adjacency matrix obtained by either keeping the input adjacency matrix A unchanged or by removing some edges from A. For all $\boldsymbol{X}$ such that $X_i^{(t)} \leq \min\{k_s, k_z\}$, we have $\Phi_1^a(\boldsymbol{X}, \bar{A}) = \Phi_2^a(\boldsymbol{X}, \bar{A})$.*

Proof. First, if A is fixed, from the forwarding computation, we know that both models satisfy (6-16) for all $\boldsymbol{X}$ bounded by $\min\{k_s, k_z\}$. Thus, we have the Lemma for A fixed.

If the edge between nodes 1 and 2 is removed, there is no message coming to node 1 and the models' outputs will always be zeros. The remaining case is only the edge between nodes 2 and 3 is removed. In that situation, at $l = 1$, there is no incoming message to node 2 (because node 1 does not send and node 3 is disconnected) and $hr_2^{(t,l=1)} = 0$. This means at $l = 2$, there is no incoming message to node 1 because $m_{21}^{(t,l=2)} = hr_2^{(t,l=1)} = 0$. As a result, the models' outputs will also always be zeros. We then have the Lemma. □

We now state the Unidentifiable Proof for the class of Node-and-Edge-perturbation:

Theorem 6-3. *For a TGNN Φ, denote $\mathcal{P} := \{(X, \bar{A}, \Phi(X, \bar{A}))|X_i \leq K\}_{X \in \mathcal{X}, \bar{A}}$, i.e. the set of Node-and-Edge-perturbation-response of Φ where $\boldsymbol{X}$ is bounded by K, and $\bar{A}$ is defined as in Lemma 6-5. Denote g an algorithm accepting $\mathcal{P}$ as inputs. For any $K > 0$ and g, there exists a Φ satisfying the two conditions in Theorem 6-1.*

Proof. We first choose k_s and k_z in Table 6-3 to K in the Theorem. We then construct Φ_1^a and Φ_2^a as described in Sect. 6.6. Denote $\mathcal{P}_1$ and $\mathcal{P}_2$ the sets of Node-and-Edge-perturbation-response of Φ_1^a and Φ_2^a, respectively. Note that, from the discussion of Transparent Models in that section, we have $\mathcal{B}_1^a$ and $\mathcal{B}_2^a$ are the Transparent Models the two models.

Given an Node-and-Edge-perturbation-response, suppose g returns either $\mathcal{B}_1^a$ or $\mathcal{B}_2^a$. From Lemma 6-5, $\mathcal{P}_1$ is the same as $\mathcal{P}_2$; therefore, the outputs of g on the 2 perturbation-response sets must be the same. Following similar arguments as in the proof of Theorem 6-1, we have Theorem 6-3. □

As in previous proofs, the proof of Theorem 6-3 also shows Node-and-Edge-perturbation cannot differentiate Φ_1^a with Φ_2^a, whose Transparent Models are two DBNs with different temporal information. Therefore, we can conclude that Node-and-Edge-perturbation cannot identify the model's components conducting the temporal messaging and aggregations.

6.7 Experiments

This section reports our experiments demonstrating the impact of unidentifiable information in the explanation tasks for GNNs and TGNNs, in both synthetic and real-world settings. In particular, our experiments aim to elaborate on the failures of Node-perturbation in identifying the correct propagating paths (Sect. 6.4) and Node-and-Edge-perturbation in capturing the temporal information carried out by the models (Sect. 6.6).

6.7.1 Synthetic Experiment on Node-perturbation

To demonstrate the unidentifiable result of Node-perturbation, we construct a synthetic node-classification task as shown in Fig. 6-9. The input is a 6-node-circle graph with one activated node. The task is to transmit that activation to the 2-hop away nodes from those activated nodes using 2 graph layers. Due to the restriction of 2 graph layers, the ground-truth explanation must be as indicated in Fig. 6-9 **(iv)**.

Node-perturbation methods are expected to fail since they cannot differentiate the contributions of the two 1-hop neighbors of the target node. We consider the state-of-the-art explanation method, the GNNExplainer [155], to elaborate that claim. Table 6-5 reports the True-positive-rate (TPR) and False-positive-rate (FPR) of the method in this synthetic task. The result not only supports our claim in the case of Node-perturbation but also in Node-and-Edge perturbation. Interestingly, a naive greedy Edge-perturbation method in which the edges are selected based on their sensitivity to the predictions can match the ground truth perfectly. These results emphasize clearly the importance of the choice of perturbations for explanations.

6.7.2 Experiments on Node-and-Edge-perturbation

We now describe our synthetic experiment showing the issue of explaining TGNNs' temporal dynamic using Node-and-Edge perturbation. The learning task is the prediction of the maximum signals observed in the opposite node in a 6-node-circle motif (Fig. 6-10). Two distinct 2-time-step TGNNs are obtained by the combinations of some separately trained neural layers so that their outputs are always identical on the same inputs. Furthermore, the constructions also make the two models have different temporal dynamics: while the first model transmits temporal information via the source node, the other relies on the whole graph for temporal transmission (Fig. 6-9). The existence of those 2 models implies that the unidentifiable situations pointed out in our theoretical analysis can occur as a result of the neural network's training process.

We further strengthen that claim with our experiments on the Shenzhen and Los Angeles traffic datasets [162]. The training task is to predict the traffic speed using past data. For each dataset, we use the same construction and training strategy as in the synthetic experiment to obtain 2 instances of TGNNs whose temporal activations are shown in Fig. 6-12. Since the outputs of the two TGNNs are also identical, we do not report the predictions for these experiments. It can be observed that the temporal dynamics learned by the TGNNs can be varied not only in terms of the number of nodes contributing to the temporal messaging but also in the range and meaning of activation values. The results further point out that faithful explanations for temporal models cannot be obtained by simply perturbing node and edge features of the input graph.

6.8 Discussion and Conclusion

This chapter studies the fundamental limit of different perturbation explanation methods in explaining black-box TGNNs. We have shown that there is key information on how the TGNNs generate their predictions that cannot be identified by some given classes of explanation methods. We now further point out several interesting implications of our theoretical results.

1. We first discuss the counterparts of Theorem 6-1 and 6-2 for GNNs. The Unidentifiable Proofs for Node-perturbation and Edge-perturbation explanation methods can be applied directly to GNNs by dropping the feedback loop of $H^{(t)}$ (Fig. 6-5). This modification will just set $H_i^{(t-1)}$ in (6-5) to zeros. Note that in GNNs, we do not have the temporal dimension in the interpretable domains, i.e. they are BNs instead of DBNs. As illustrations, Fig. 6-13a and 6-14 show the components of the proofs for Node-perturbation and Edge-perturbation in GNNs.

2. Theorem 6-3 for GNNs? Our proof cannot readily apply to the case of GNNs because the two constructions will have the same Transparent Model, i.e. $\mathcal{I}(\Phi_1^a) = \mathcal{I}(\Phi_2^a)$.

3. Second, we want to discuss the practical models such that the situation in our Unidentifiable Proofs can occur. As our analysis uses the most basic constructions of the TGNNs, our Unidentifiable Proofs in Sects. 6.4, 6.5 and 6.6 are applicable to all versions of

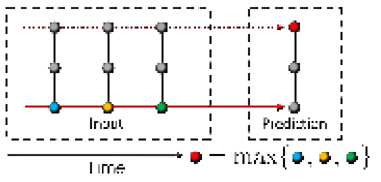
the TGNNs found in [162, 91, 92]. As the base GNN can be considered as TGNN with zero temporal feedback (see Fig. 6-5), Our results in Sects. 6.4 and 6.5 are also applicable to many modern variants of GNNs including GCN [71], GraphSage [55] and GAT [137].

4. Regarding the usage of other interpretable domains and Transparent Models (not DBNs), we can see that the Unidentifiable results can be obtained with other interpretable domains as long as (i) the Transparent Models of the constructed models can be identified (similar to Lemma 6-1 and 6-4) and (ii) they contain meaningful information that helps establish the unidentifiable information. While condition (i) requires the domain to have strong expressive power, condition (ii) requires the domain's members to be somewhat interpretable. We find DBN is a balanced choice for the analysis of TGNN.

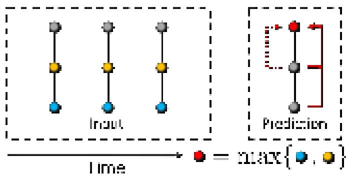
5. We now discuss the implication of Theorem 6-1 and 6-2 about the reliability of existing explanation methods for GNNs. Existing explanation methods have been successfully identifying many important features contributing to the predictions; however, the results are limited. Our results establish a fundamental limit of perturbation-based explanation methods. For example, Theorem 6-1 implies explanations obtained by only perturbing nodes cannot reliably inform us of the paths determining the predictions. For the case of the two constructed Φ_1^v and Φ_2^v, the contributions of node 2 and node 4 will always be considered equal by all Node-perturbation methods. This means both will be included or discarded by the explainers, even when the actual messages are only transmitted through one of them. Thus, Node-perturbation methods are bound to commit false positives or false negatives. This claim is further supported by our experiments in Fig. 6-9.

6. Finally, we want to discuss the implication of Theorem 6-3 about the design of future explanation methods for TGNNs. Even though the Theorem states that the Node-and-Edge perturbation methods cannot identify the temporal component of the model, it does not mean there is nothing we can do to tackle this challenging problem. Careful readers might realize that one key aspect of our proof is based on the fact that removing an edge in the

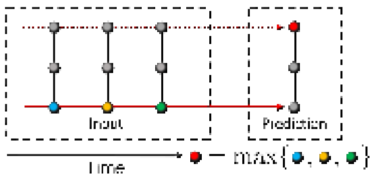
input graph will disconnect that connection at all rounds of temporal computations. If there is a mechanism to remove edge only at some temporal computations, it is possible to differentiate Φ_1^a from Φ_2^a, which is crucial to identify whether node 1 or node 3 conducts the temporal aggregation. In other words, *temporal perturbation* might be something we need to explain TGNNs more faithfully.



(a) Multi-path aggregation task.



(b) Multi-node aggregation task.

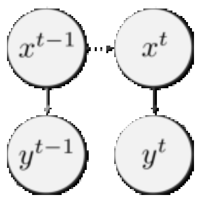


(c) Temporal aggregation task

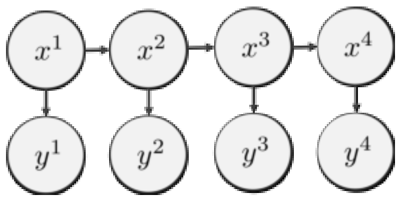
Figure 6-1. Tasks for the class of (a) Node-perturbation, (b) Edge-perturbation, and (c) Node-and-Edge-perturbation explanation methods. The dash arrows and the dotted arrows show different internal computations that the model can carry out. Our Unidentifiable Proofs show that the corresponding explanation methods cannot differentiate the computations; thus, cannot identify/explain those internal dynamics.

Table 6-1. Summary of perturbation methods used by explainers and the scope of our results.

	Node	Edge	Sect.6.4	Sect.6.5	Sect.6.6
GNNExplainer [155]	*	*			*
PGExplainer [87]		*		*	*
GraphLIME [61]	*		*		*
PGMExplainer [140]	*		*		*
RelEx [161]		*		*	*
GraphSVX [33]	*		*		*
ZORRO [45]	*		*		*



(a) A 2TBN.



(b) The unroll BNs.

Figure 6-2. An example of a DBN and its unroll BN. The intra-slice connections are solid and the inter-slice connections are dashed. The brighter line can be omitted as it can be inferred from other edges.

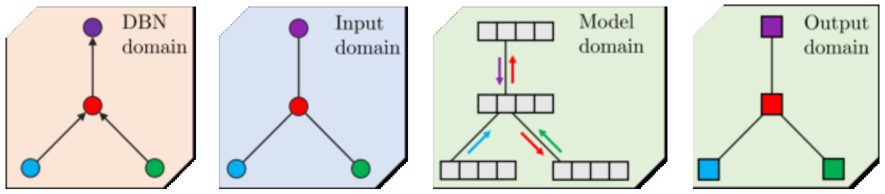


Figure 6-3. The association among variables of the explanation DBN, the input nodes and the messages in the TGNN: Components of the same color are associated with each other.

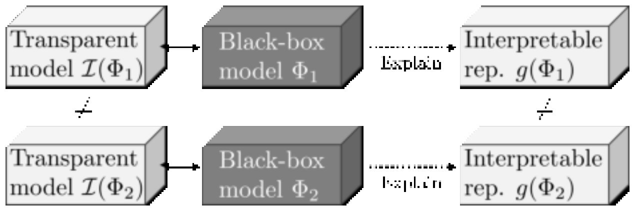

Transparent model $\mathcal{I}(\Phi_1)$
Black-box model Φ_1
Explain
Interpretable rep. $g(\Phi_1)$
Transparent model $\mathcal{I}(\Phi_2)$
Black-box model Φ_2
Explain
Interpretable rep. $g(\Phi_2)$

Figure 6-4. Necessary conditions for explanations under the Existence assumption.

Table 6-2. The constant features of the TGNNs in G_v's proof.

Node	1		2		3		4	
Hidden features	hs_1	hz_1	hs_2	hz_2	hs_3	hz_3	hs_4	hz_4
TGNN Φ_1^v	k_s	0	0	k_z	0	k_z	k_s	k_z
TGNN Φ_2^v	k_s	0	k_s	k_z	0	k_z	0	k_z

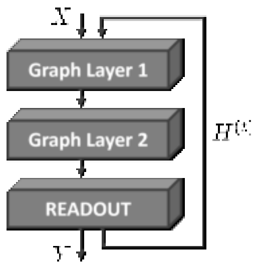


Figure 6-5. The general TGNN model.

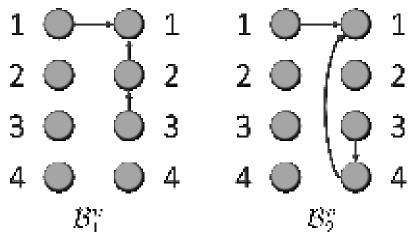


Figure 6-6. The DBNs explaining the two T-GNNs Φ_1^v and Φ_2^v.

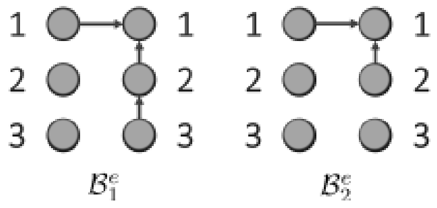


Figure 6-7. The DBNs for Unidentifiable Proof of $\mathcal{G}_e$.

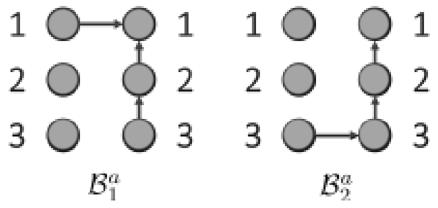


Figure 6-8. The DBNs for Unidentifiable Proof of $\mathcal{G}_a$.

Table 6-3. The constant features of the TGNNs in G_a's proof.

Node	1			2			3		
Features	hs_1	hz_1	ho_1	hs_2	hz_2	ho_2	hs_3	hz_3	ho_3
TGNN Φ_1^a	k_s	0	0	0	k_z	0	0	k_z	0
TGNN Φ_2^a	k_s	0	0	0	k_z	k_z	k_s	k_z	k_z

Table 6-4. Hidden features of the TGNN Φ_2^a for input $X^{(t=1)} = [\alpha_1, \alpha_2, \alpha_3]$ and $X^{(t=2)} = [\beta_1, \beta_2, \beta_3]$. $\gamma_3 = \max\{\alpha_3, \beta_3\}$ and $\gamma_2 = \max\{\gamma_3, \beta_2\}$

Variable		Input	Layer 1	Layer 2
$t = 1$	m_{i*}		$0, \alpha_2, \alpha_3$	$0, \alpha_3, \alpha_2$
$H^{(t)} = [0,0,0]$	hr_i	$\alpha_1, \alpha_2, \alpha_3$	$\alpha_2, \alpha_3, \alpha_2$	$\alpha_3, \alpha_2, \alpha_3$
$t = 2$	m_{i*}		$0, \beta_2, \gamma_3$	$0, \gamma_3, \gamma_2$
$H^{(t)} = [0,0,\alpha_3]$	hr_i	$\beta_1, \beta_2, \beta_3$	$\beta_2, \gamma_3, \beta_2$	$\gamma_3, \gamma_2, \gamma_3$

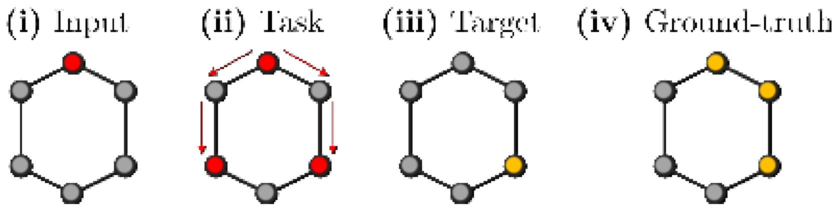


Figure 6-9. The synthetic static task demonstrating the failure of node-perturbation explanation methods.

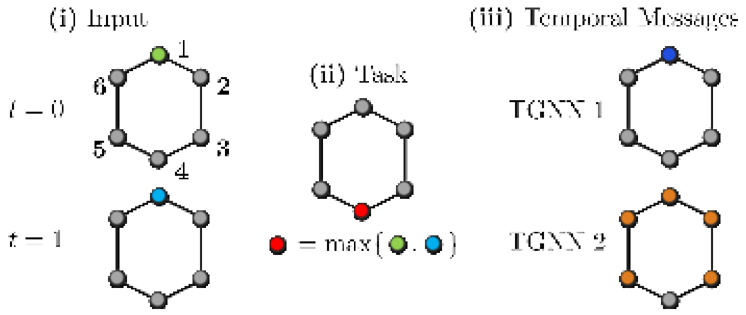


Figure 6-10. The synthetic temporal task demonstrating the failure of node-and-edge perturbation-based explanation methods.

Table 6-5. GNNExplainer fails to detect the ground-truth explanations of the synthetic task (Fig. 6-9) when using Node-perturbation (Node) and Node-and-Edge perturbation (All).

Method	Return Edge		Return Node	
	TPR	FPR	TPR	FPR
Naive Edge	1.00 ± 0.00	0.00 ± 0.00	1.00 ± 0.00	0.00 ± 0.00
GNNExplainer (Node)	1.00 ± 0.00	0.50 ± 0.00	0.61 ± 0.20	0.39 ± 0.20
GNNExplainer (All)	0.50 ± 0.18	0.25 ± 0.09	0.58 ± 0.20	0.42 ± 0.20

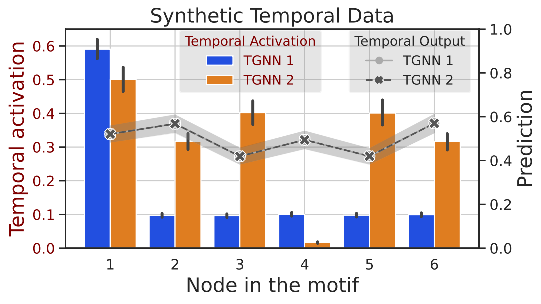


Figure 6-11. The temporal activations of TGNNs in synthetic temporal task demonstrating the failure of Node-and-edge perturbation-based explanation methods.

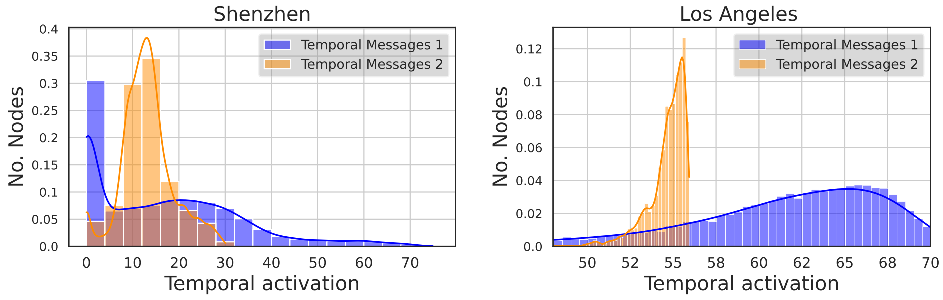
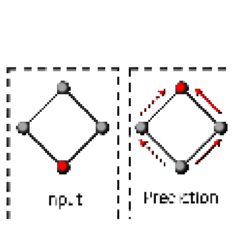
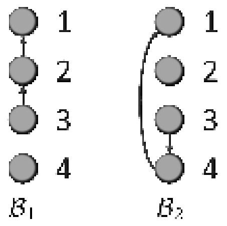


Figure 6-12. The real-world temporal messages demonstrating the failure of Node-and-edge perturbation-based explanation methods. Note that the outputs of the two TGNNs in both experiments are identical regardless of the perturbations.



(a) The training task.



(b) The BNs.

Figure 6-13. The components for Node-perturbation Unidentifiable Proof of GNNs.

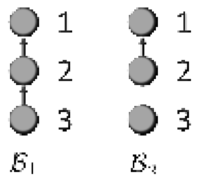


Figure 6-14. The BNs for Edge-perturbation Unidentifiable Proof of GNNs.

APPENDIX A
EXPERIMENTS OF C-EVAL ON CIFAR10

Besides MNIST and Caltech101, we also conduct experiment on the small color image dataset CIFAR10 [74]. The distributions of *c*-Eval on 500 images of the dataset and some examples are shown in Figs. A-1 and B-2. The experimental parameters model and the segmentation procedure are similar to that on Caltech101 dataset. The classifier model in this experiment is the adaptation of VGG on CIFAR10 [84]. Results in Fig. B-2 suggest a relative ranking in performance of studied explainers. We do not include this result in the main manuscript because, to the extend of our knowledge, there is no evaluation on performance of explanations on this dataset. However, we think that this result can serve as a reference for our MNIST and Caltech101 experiments, which is described in Subsection 2.5.5.

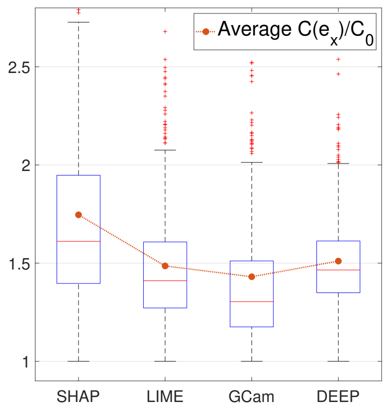


Figure A-1. Distributions of *c*-Eval on CIFAR10 dataset.

APPENDIX B
EXAMPLES OF EXPLANATIONS OF MNIST, CIFAR10 AND CALTECH101

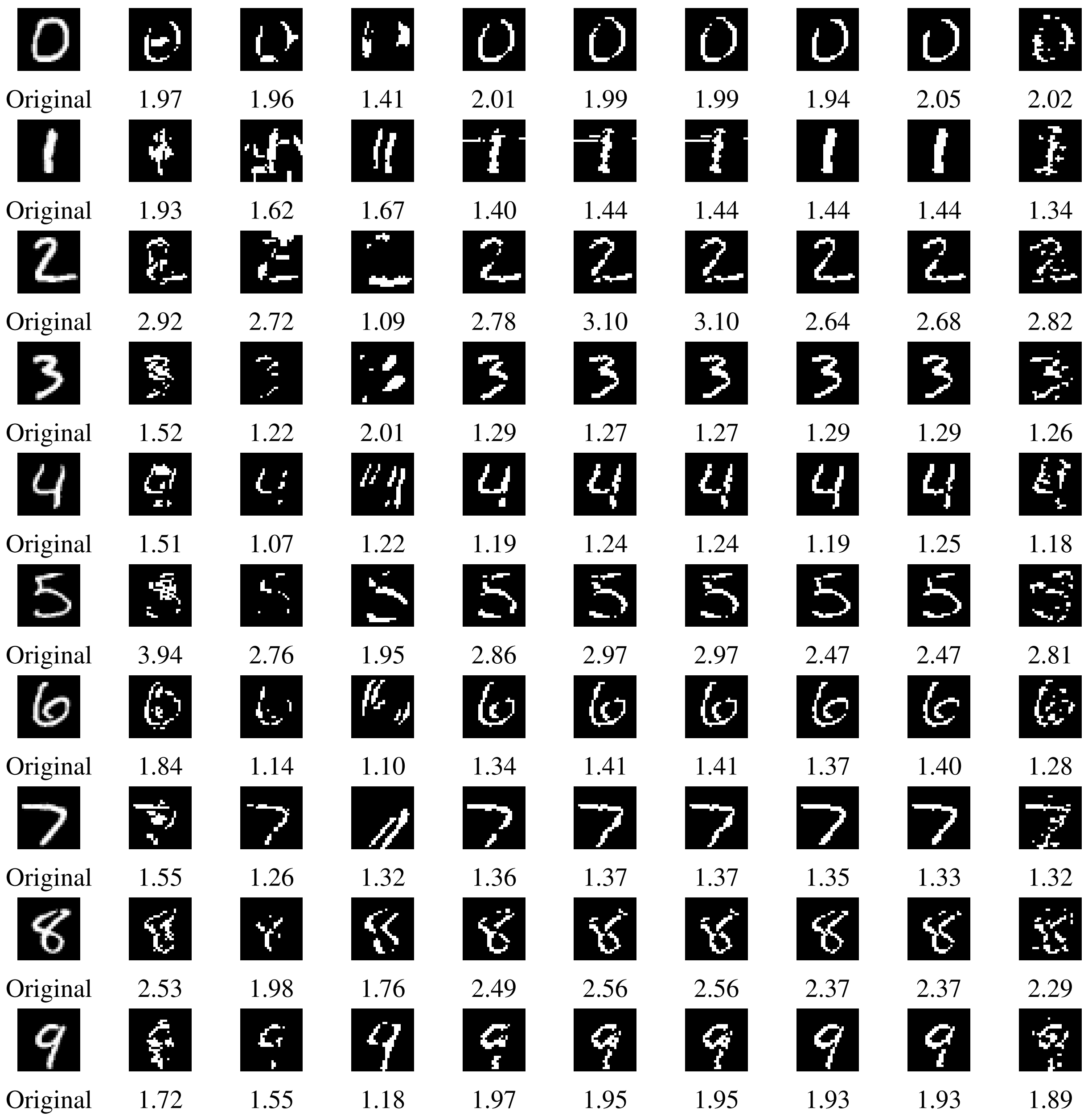

Original 1.97 1.96 1.41 2.01 1.99 1.99 1.94 2.05 2.02
Original 1.93 1.62 1.67 1.40 1.44 1.44 1.44 1.44 1.34
Original 2.92 2.72 1.09 2.78 3.10 3.10 2.64 2.68 2.82
Original 1.52 1.22 2.01 1.29 1.27 1.27 1.29 1.29 1.26
Original 1.51 1.07 1.22 1.19 1.24 1.24 1.19 1.25 1.18
Original 3.94 2.76 1.95 2.86 2.97 2.97 2.47 2.47 2.81
Original 1.84 1.14 1.10 1.34 1.41 1.41 1.37 1.40 1.28
Original 1.55 1.26 1.32 1.36 1.37 1.37 1.35 1.33 1.32
Original 2.53 1.98 1.76 2.49 2.56 2.56 2.37 2.37 2.29
Original 1.72 1.55 1.18 1.97 1.95 1.95 1.93 1.93 1.89

Figure B-1. Some examples of explanations and c-Eval on MNIST. The explainers from left to right: SHAP, LIME, GCam, DeepLIFT, Integrated Gradient with 5 and 10 interpolations, Guided Backpropagation, and Gradient. The number associated with each figure is the ratio $c_{f,\boldsymbol{x}}(e_{\boldsymbol{x}})/c_{f,\boldsymbol{x}}(\emptyset)$.

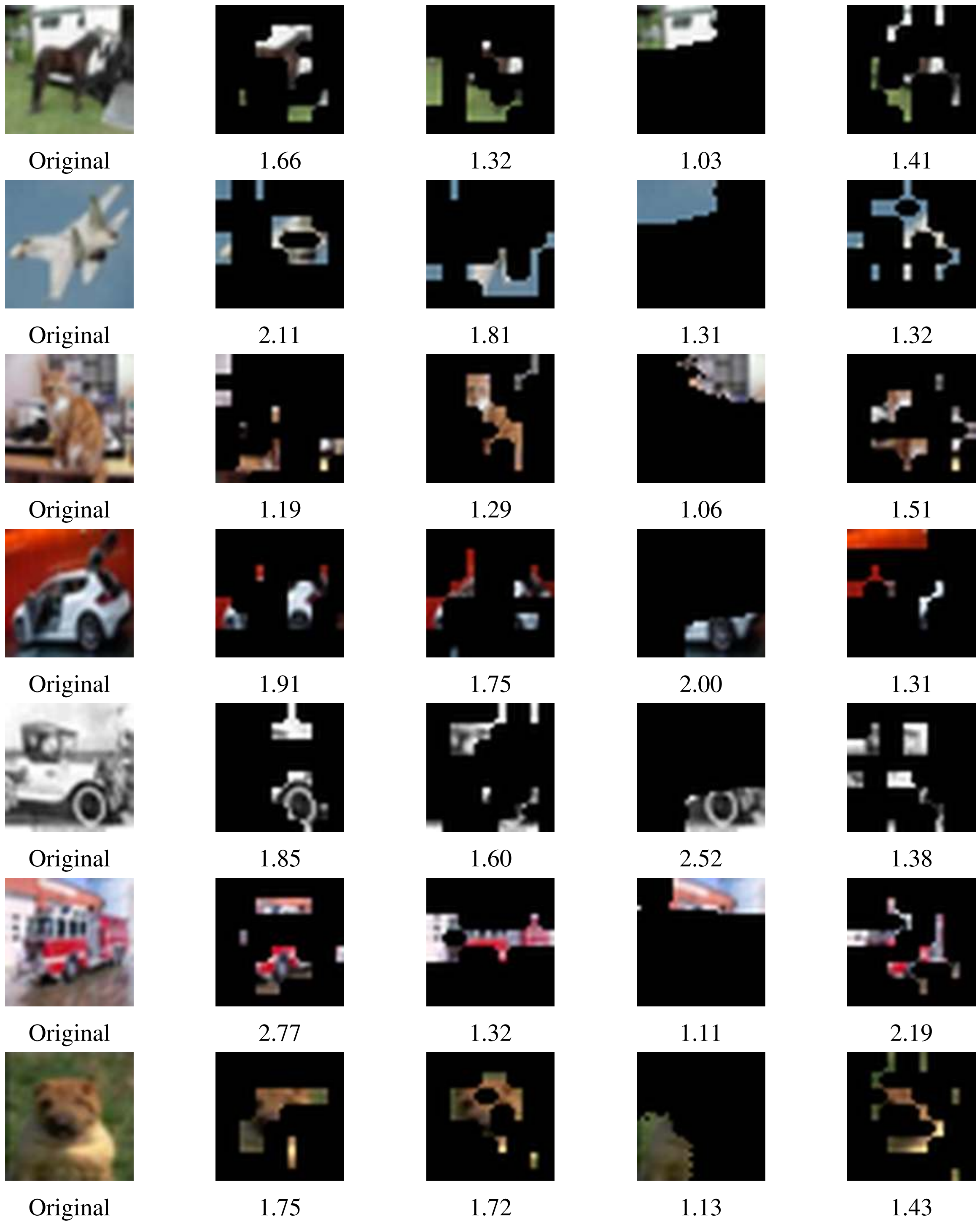


Figure B-2. Some examples of Explanations and c-Eval on CIFAR10. The explainers from left to right: SHAP, LIME, GCam and DeepLIFT. The number associated with each figure is the ratio $c_{f,\boldsymbol{x}}(e_{\boldsymbol{x}})/c_{f,\boldsymbol{x}}(\emptyset)$. We observe that most explanations which capture the signature components of the images have relatively high c-Eval.

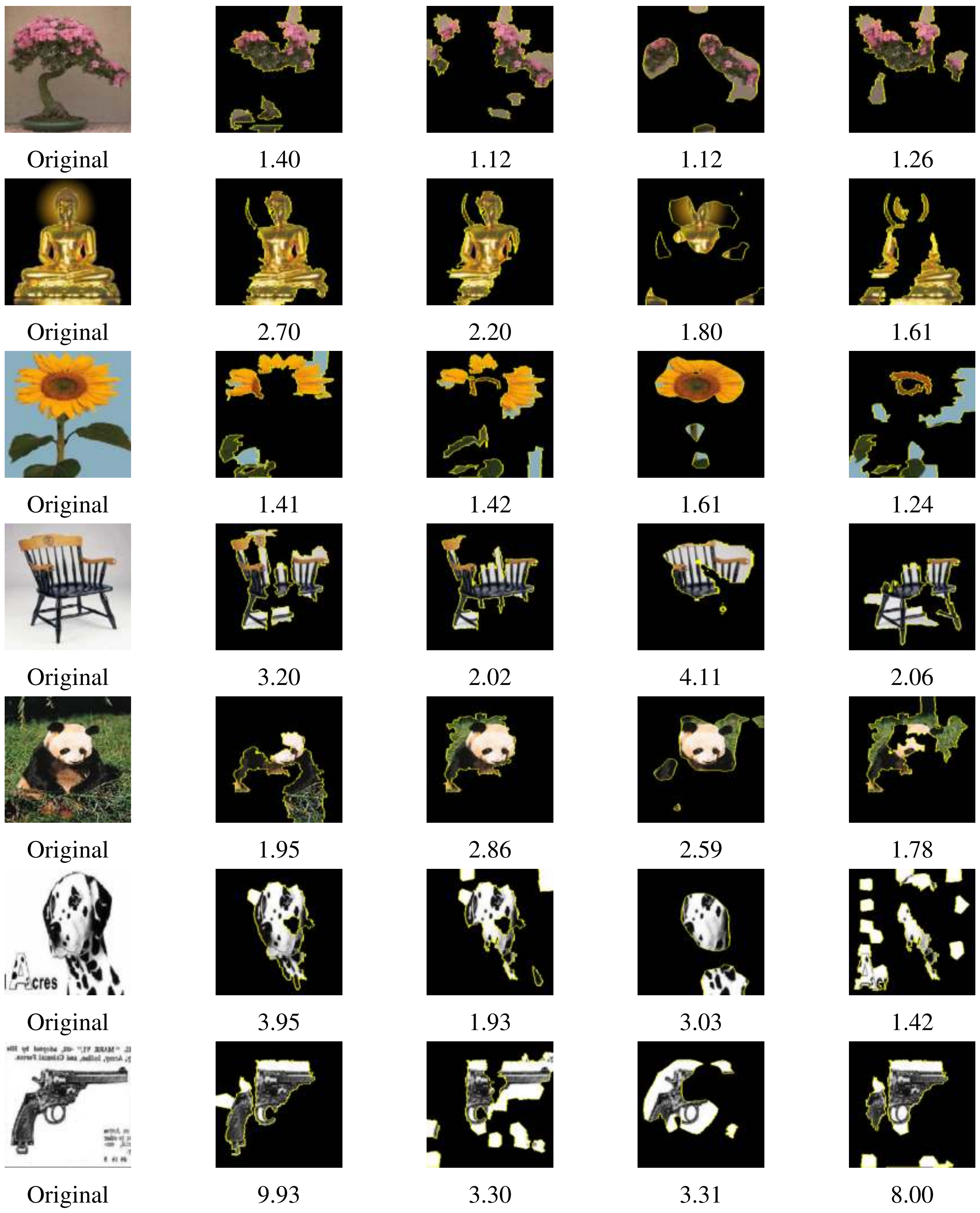


Figure B-3. Some examples of Explanations and c-Eval on Caltech101. The explainers from left to right: SHAP, LIME, GCam and DeepLIFT. The number associated with each figure is the ratio $c_{f,\boldsymbol{x}}(e_{\boldsymbol{x}})/c_{f,\boldsymbol{x}}(\emptyset)$. We observe that most explanations with high c-Eval contain important features of the input images.

APPENDIX C
VISUALIZATIONS OF SYNTHETIC DATA AND EXPLANATIONS GENERATED WITH EMAP

This Appendix provides some visualizations of our synthetic data (in experiments of Table 3-3) and explanations generated with or without EMaP. The explanations of EMaP shown in this Appendix are those that appeared in the experiments reported in Section 3.6.

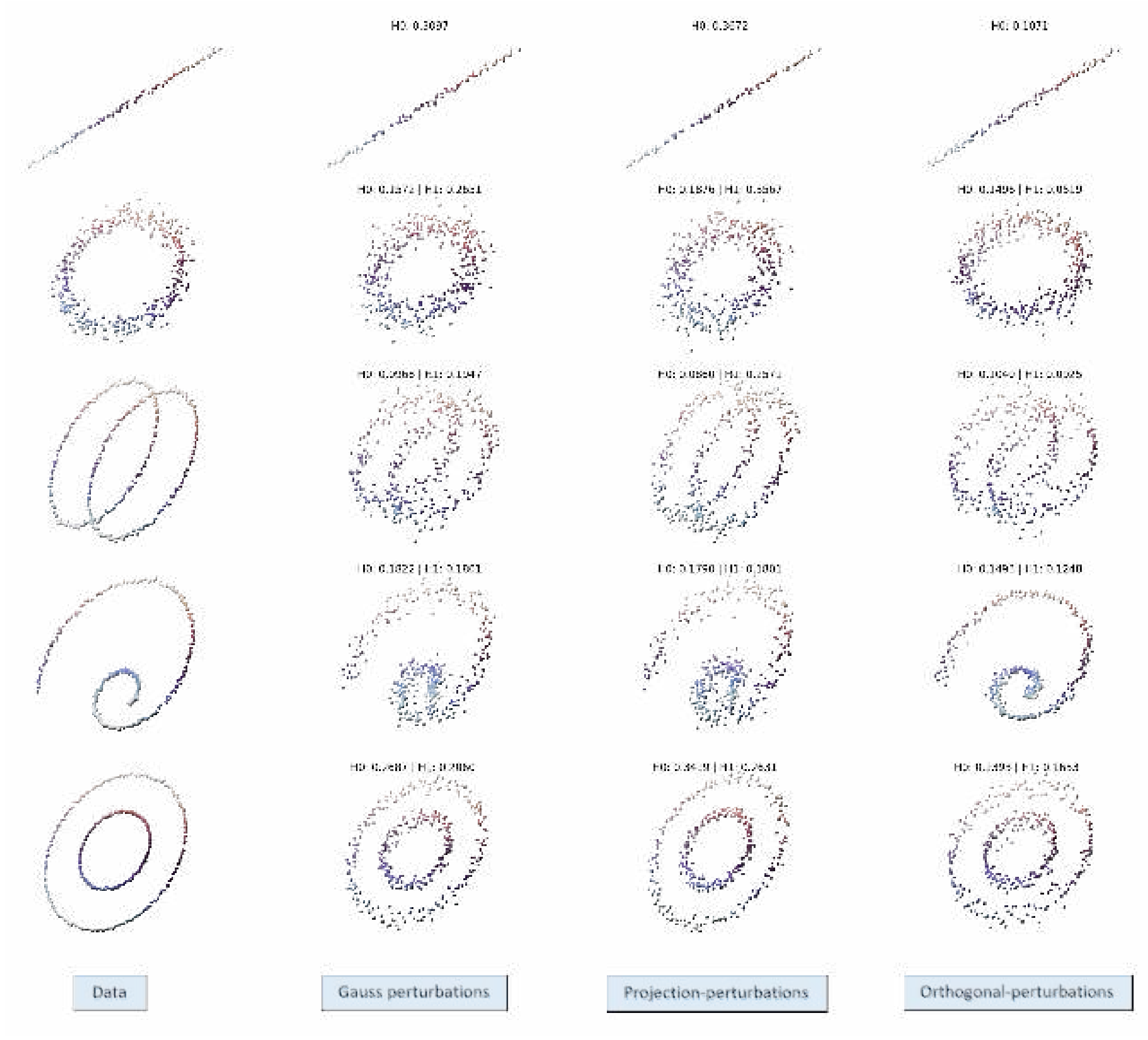
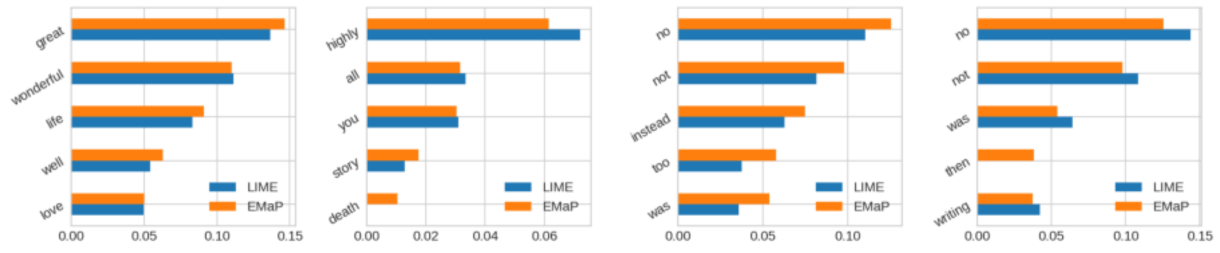


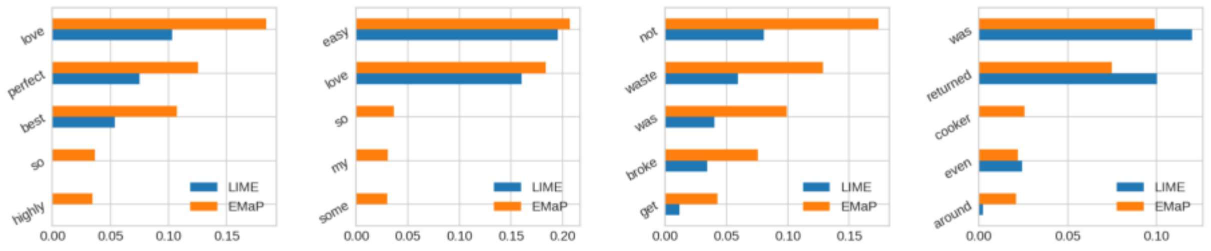
Figure C-1. Visualizations of some synthetic data in experiments of Table 3-3.

In Fig. C-1, we visualize the synthetic data of different shapes and their perturbations in three dimensions. We also report H_0 and H_1 Bottleneck distances between the perturbations and the original data in that figure.



Positive Positive Negative Negative

Figure C-2. Visualization of EMaP-LIME explanations of the Multi-polarity-books review datasets [15].



Positive Positive Negative Negative

Figure C-3. Visualization of EMaP-LIME explanations of the Multi-polarity-kitchen review datasets [15].

Fig. C-2 and C-3 compare the actual explanations returned by LIME and EMaP-LIME (EMaP) in the books' reviews in the Multi-Domain Sentiment datasets. We can see that the weights of features included in the explanations are quite similar between the two methods.

Fig C-4 shows the explanations of LIME with EMaP for all classes in the MNIST dataset. The red (blue) areas mean, if the features in those areas are unchanged (changed), the prediction for that class will be stronger (weaker).

Finally, Fig. C-5 and C-6 provide some explanations of EMaP along with explanations of other methods in MNIST and Fashion-MNIST, respectively. The color code also has a similar meaning to that of Fig C-4.

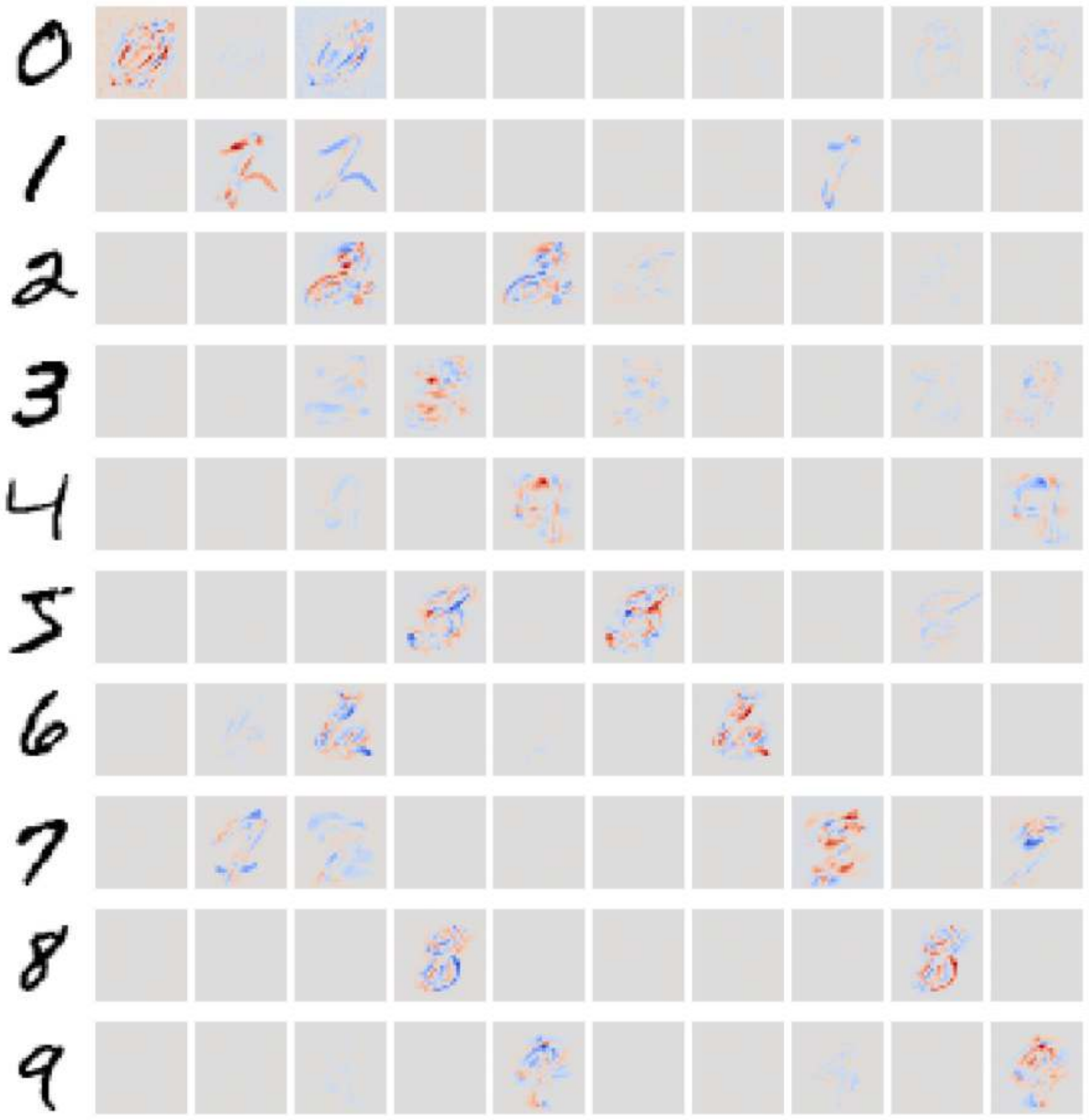


Figure C-4. Visualization of EMaP-LIME explanations of the MNIST dataset. The column indicates the class label to be explained.

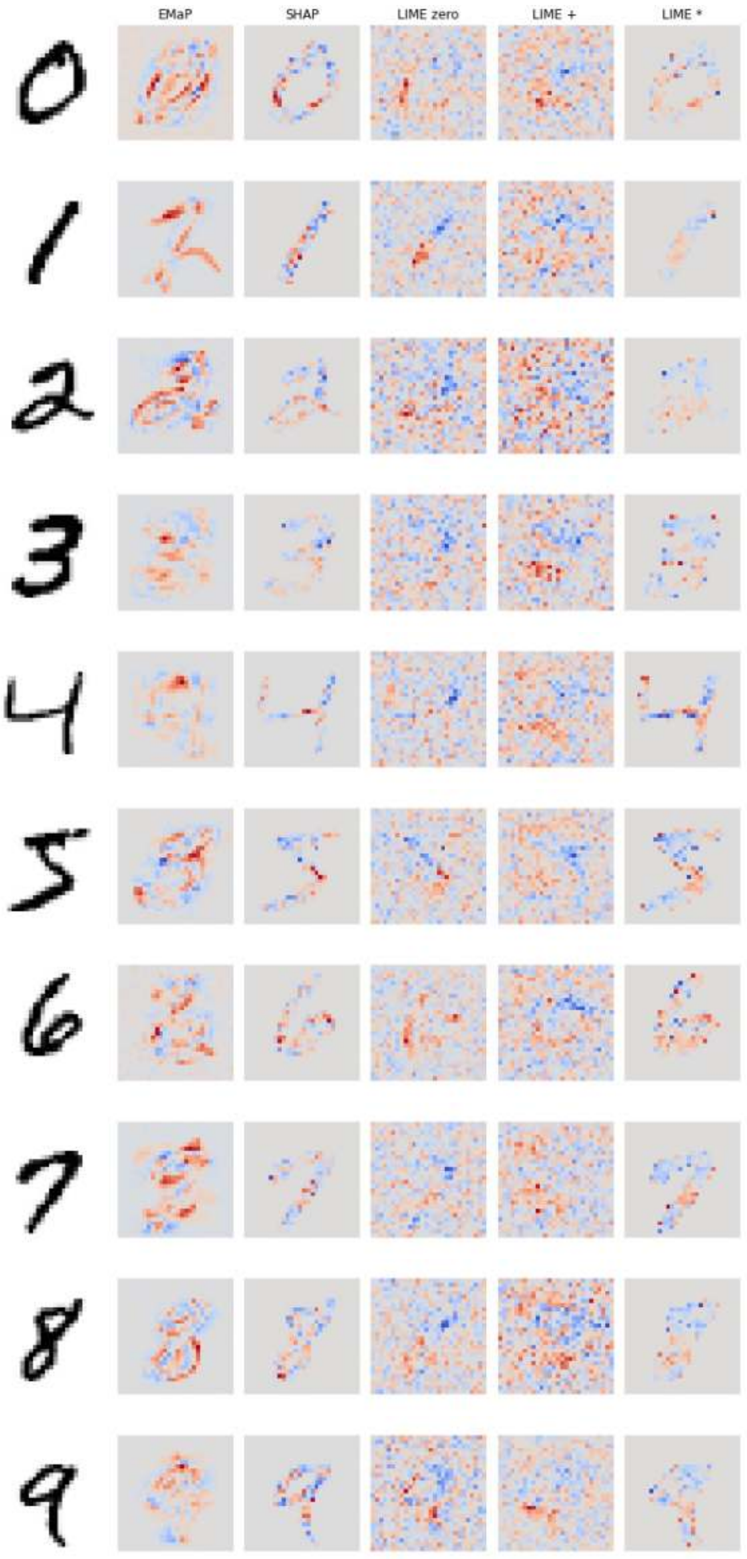


Figure C-5. Examples of explanations in MNIST. Modifying the red-est (blue-est) area would negate (strengthen) the original prediction.

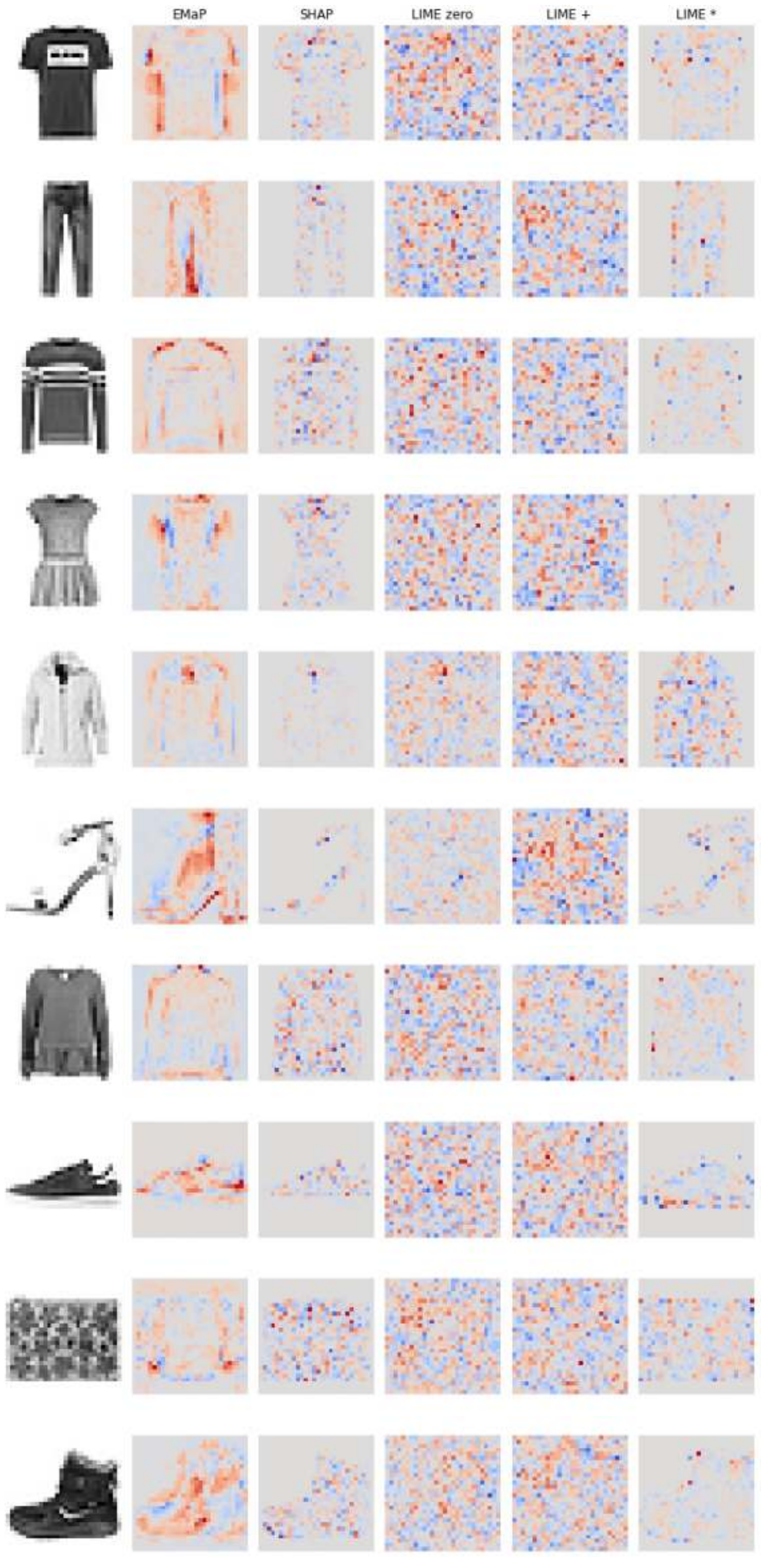


Figure C-6. Examples of explanations in Fashion-MNIST. Modifying the red-est (blue-est) area would negate (strengthen) the original prediction.

APPENDIX D
ADDITIVE FEATURE ATTRIBUTION METHODS UNIFY EXISTING EXPLAINERS FOR GNNS

In this section, we analyze the vanilla gradient-based explainers and GNNExplainer [155] under the explanation model framework. Our aim is to bring a clearer view on explanation models and show that the class of *additive feature attribution methods* introduced in [86] fully captures current explanation methods for GNNs.

Gradient-based explainers: The gradient-based approach has been one of the most popular explanation methods for conventional neural networks [121, 122, 130, 9, 128, 123]. Some of those recently have been analyzed in the context of GNNs [106], including Gradient-Based Saliency maps [123], Grad-CAM [121] and Excitation Back-Propagation [64]. All of these methods rely on back-propagating the GNN and assigning important scores on explained features. Here, we consider the simplest gradient-based explanation method in which the score of each feature is associated with the gradient of the GNN's loss function with respect to that feature. The top-M nodes with the largest sum of their features' scores are selected as the explanation.

The proof that this explanation method falls into the class of *additive feature attribution methods* is quite straightforward. Here, the interpretable domain $\mathcal{E}$ is a set of linear functions of the form $\zeta(z) = \phi_0 + \sum_{i=1}^{|V|} \phi_i z_i$, where $z_i \in \{0,1\}$ is a binary variable representing the selection of node i in the explanation and $\phi_i \in \mathbb{R}$ is the associated sum gradients in node i. The objective R and the feasible sets $\mathcal{C}$ can be chosen as follows:

$$R_{\Phi,t}(\zeta) = \sum_{v\in V, z_v=1} \left(\sum_{x\in X_v} \frac{\partial \Phi_t}{\partial x} \right), \quad \mathcal{C} = \left\{ \zeta(z) : \sum_{i=1}^{|V|} z_i = M \right\}. \tag{D-1}$$

To show other gradient-based explanation methods [106] belong to the class of *additive feature attribution methods*, we just need to adjust the choice of the weights ϕ_i in the above expression.

GNNExplainer: To generate a subgraph $S = (V_S, E_S)$ with at most M edges explaining a target prediction, GNNExplainer is formulated based on the following optimization:

$$\max_{S\subseteq G} \mathrm{I}(Y;S) = \max_{S\subseteq G} \left(\mathrm{H}(Y) - \mathrm{H}(Y|\mathcal{G} \equiv S) \right), \quad |E_S| \le M, \tag{D-2}$$

where I and H are the mutual information function and the entropy function respectively. S is a random subgraph of G and Y is a random variable with the probability that the target prediction belongs to each of $\mathcal{K}$ classes. In GNNExplainer, the distribution of Y is obtained from the soft output of the GNN at different input settings. The condition $\mathcal{G} \equiv S$ indicates that the realization of $\mathcal{G}$ must be consistent with the realization of subgraph S. The intuition on this objective is that, if knowing the information in the subgraph S reduces the uncertainty of Y significantly, the subgraph S is a good explanation for the target prediction.

To avoid the complexity of directly solving (D-2), GNNExplainer relaxes the integer constraint on the entries of the input adjacency matrix A and searches for an optimal fractional adjacency matrix $\tilde{A} \in [0,1]^{|V|\times|V|}$. The distribution of S is then defined by setting $P(S) = \prod_{(i,j)\in S} \tilde{A}_{ij}$. After that, by approximating $\mathrm{H}(Y|\mathcal{G} \equiv S) \approx \mathrm{H}(Y|\mathcal{G} \equiv \mathbb{E}[S]) = \mathrm{H}(Y|\tilde{A})$, the explainer uses mean-field optimization to solve for $\min_{\tilde{A}} \mathrm{H}(Y|\mathcal{G} \equiv \mathbb{E}[S])$. Finally, the explanation S is selected based on the values of entries in the solution $\tilde{A}$. Note that in order to evaluate $\mathrm{H}(Y|\tilde{A})$, the explained model must be able to compute the prediction at fractional adjacency matrix input. However, in many current GNN architectures such as GraphSage [55] and GIN-Graph [150], the information of the adjacency matrix is used to compute the aggregated discrete sum of features and operations on float inputs of the adjacency matrix are not well-defined. Thus, GNNExplainer would fail to explain the predictions of those models.

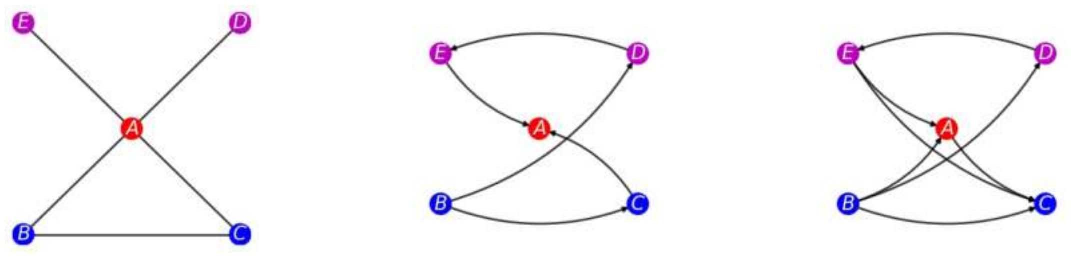
If we do not allow the entries of $\tilde{A}$ to receive fractional value, the optimization problem of GNNExplainer can be formulated under the *additive feature attribution methods* by setting $\zeta(\tilde{A}) = \phi_0 + \sum_{i=1}^{|V|}\sum_{j=1}^{|V|} \phi_{ij}\tilde{A}_{ij}$ where $\tilde{A}_{ij} \in \{0,1\}$. The objective R and the feasible set $\mathcal{C}$ are specified as follows:

$$R_{\Phi,t}(\zeta) = -\mathrm{H}(\Phi(\mathcal{G})_t|\tilde{A}), \quad \mathcal{C} = \left\{ \zeta(\tilde{A}) : \sum_{i=1}^{|V|}\sum_{j=1}^{|V|} \tilde{A}_{ij} \leq 2M, \tilde{A}_{ij} = \tilde{A}_{ji} \right\}. \tag{D-3}$$

The weights ϕ_{ij} of explanation model ζ are chosen based on the solution in maximizing $R_{\Phi,t}(\zeta)$.

APPENDIX E
EXAMPLE OF PGM-EXPLAINER WITH AND WITHOUT *NO-CHILD* CONSTRAINT

In Fig. E-1, we provide an example illustrating the impact of the *no-child* constraint (5-3) onto the PGM explanation. This experiment setup is the same as that in the experiment of Fig. 5-1. Here, we use PGM-Explainer to explain the prediction on the role ”blue” of node C shown in Fig.E-1a. Fig.E-1b and Fig.E-1c are the resulting explanations without and with the *no-child* constraint.



(a) Ground-truth motif. (b) Without *no-child* constraint. (c) With *no-child* constraint.

Figure E-1. Example of PGM-explanation with and without *no-child* constraint. **(a)** The ground-truth motif containing the target prediction to be explained, which is the prediction on node C. **(b)** A PGM-explanation in a form of Bayesian network without the *no-child* constraint. **(c)** A PGM-explanation in a form of Bayesian network with the *no-child* constraint.

As can be seen in this experiment, imposing the constraint does not change the set of nodes identified by the explainer. However, the constraint changes the edges in the Bayesian network. In general, without the constraint, we will obtain a more simple model: fewer edges, less number of parameters, and more independent assertions. While this network is faithful to the data, it might be misleading to non-experts. In fact, the Markov-blanket of C in this graph is $\{A,B,E\}$. Notice that E is in the Markov blanket even when there is no chain of one-direction arrows connecting C and E. By imposing the *no-child* constraint, the resulting network here reverses the arrow $C \rightarrow A$ and adds two additional arrows $B \rightarrow A$ and $E \rightarrow C$. These arrows are included to guarantee that the resulting network has enough power to represent the distribution generating the data. Even though this network is more complex, it is intuitive in the sense that we can directly point out the Markov-blanket of C, which is its parents. Furthermore, if the Bayesian network is stored in factorization form, it is also much more convenient to perform inference queries on the target variable C when it has no child. To see this, consider the factorized forms of the networks in Fig. E-1b and E-1c, which are $P(B)P(C|B)P(D|B)P(E|D)P(A|C,E)$ and $P(B)P(D|B)P(E|D)P(A|B,E)P(C|A,B,E)$ respectively. When storing the networks, we only store each conditional probability. Thus, answering inference queries regarding the target variable C is much more convenient in the latter network as all required information is stored in $P(C|A,B,E)$. To sum up, the *no-child* constraint might increase the complexity of the explanation model; however, it offers better intuition and is more convenient in performing inference on the target of the explanation.

APPENDIX F
PROBABILISTIC GRAPHICAL MODEL PRELIMINARIES

For completeness of this work, we provide some formal definitions of PGM-related concepts in this section. For more and complete information about PGM, readers are referred to [73].

F.1 I-Map and Perfect Map

Given a distribution P, we denote $\mathcal{I}(P)$ to be the set of independence assertions of the form $(\boldsymbol{x} \perp\!\!\!\perp \boldsymbol{y}|\boldsymbol{z})$ that hold in P. Similarly, given a Bayesian network $\mathcal{B}$, $\mathcal{I}(\mathcal{B})$ is the set of independence assertions made by $\mathcal{B}$. For $\mathcal{B}$ to faithfully represent P, $\mathcal{B}$ must not contain any independence assertions that are not made by P, i.e. $\mathcal{I}(\mathcal{B}) \subseteq \mathcal{I}(P)$. In that case, we say $\mathcal{B}$ is an I-map for P.

In the context of PGM, we normally want to find the simplest model $\mathcal{B}$ that is able to represent P. Roughly speaking, the larger the set $\mathcal{I}(\mathcal{B})$, the simpler the model $\mathcal{B}$ since we would need less number of parameters to determine the network. This leads us to the following definition of minimal I-map.

Definition 6. *A Bayesian network $\mathcal{B}$ is a minimal I-map for P if it is an I-map for P, and if the removal of even a single edge from $\mathcal{B}$ renders it not an I-map.*

Therefore, to capture the independence structure in P, a standard solution is to search for minimal I-maps of P. However, as shown by [73], even if a network $\mathcal{B}$ is a minimal I-map, it may not include many important independence assertions in P and fails to capture the distribution's structure. Alternatively, we aim to find a graph, called a perfect map, that precisely captures P.

Definition 7. *A Bayesian network $\mathcal{B}$ is a perfect map for P if $\mathcal{I}(\mathcal{B}) = \mathcal{I}(P)$.*

Unfortunately, not every distribution P has a perfect map. In the scope of this work, we consider the case when the perfect map for the distribution generating the data exists, i.e. the class of Bayesian networks has the power to represent the distribution precisely.

F.2 Markov-Blanket

In the variable selection step of PGM-Explainer (Section 5.2.2), we solve for the set $\boldsymbol{U}(\boldsymbol{t})$ that guarantees to contain the Markov-blanket of the target variable in case the perfect map exists. Here, we provide the formal definition of the Markov-blanket in a distribution.

Definition 8. *Given a distribution P on a set of random variables $\mathcal{X}$, a set $\boldsymbol{S}$ is a Markov-blanket of $\boldsymbol{x} \in \mathcal{X}$ if $\boldsymbol{x} \notin \boldsymbol{S}$ and if $\boldsymbol{S}$ is a minimal set such that $(\boldsymbol{x} \perp\!\!\!\perp \mathcal{X} \setminus (\{\boldsymbol{x}\} \cup \boldsymbol{S}) \,|\boldsymbol{S}) \in \mathcal{I}(P)$. We denote this set* $\mathrm{MB}_{\mathcal{P}}(\boldsymbol{x})$.

If the perfect map $\mathcal{B}^*$ for P exists, then $\mathrm{MB}_{\mathcal{P}}(\boldsymbol{x})$ is simply $\boldsymbol{x}$'s parents, $\boldsymbol{x}$'s children, and $\boldsymbol{x}$'s children's other parents in $\mathcal{B}^*$ [73].

F.3 Structure Learning and Bayesian Information Criterion (BIC) Score

The task of learning a graph structure from data is called structure learning. Algorithms for structure learning can be roughly classified into two categories: score-based and constraint-based. The score-based algorithms associate a score to each candidate structure with respect to the training data and then search for a high-scoring structure. The constraint-based algorithms look for dependencies and conditional dependencies in the data and construct the structure accordingly.

In a score-based approach, the selection of a scoring function is critical. A natural choice of the score function is the likelihood $l(\hat{\theta}_{\mathcal{B}} : \mathcal{D})$, where $\mathcal{D}$ is the data, $\mathcal{B}$ is the graphical model of consideration, $\hat{\theta}_{\mathcal{B}}$ is the maximum likelihood parameters for $\mathcal{B}$ and l is the logarithm of the likelihood function. However, as pointed out in [73], the likelihood score overfits the training data and returns overly complex structure. An alternative solution is the Bayesian score

$$\mathrm{score}_B(\mathcal{B} : \mathcal{D}) := \log P(\mathcal{D}|\mathcal{B}) + \log P(\mathcal{B}), \tag{F-1}$$

where $P(\mathcal{B})$ is the prior over structures. However, this prior is almost insignificant compared to the marginal likelihood $P(\mathcal{D}|\mathcal{B})$, which can be computed as:

$$P(\mathcal{D}|\mathcal{B}) = \int P(\mathcal{D}|\theta_{\mathcal{B}}, \mathcal{B}) P(\theta_{\mathcal{B}}|\mathcal{B}) d\theta_{\mathcal{B}}. \tag{F-2}$$

It can be shown that if we use a Dirichlet parameter prior for all parameters in $\mathcal{B}$, then when the number of samples n is sufficiently large, we have

$$\log P(\mathcal{D}|\mathcal{B}) = l(\hat{\theta}_{\mathcal{B}} : \mathcal{D}) - \frac{\log n}{2}\mathrm{Dim}[\mathcal{B}] + O(1). \tag{F-3}$$

Thus, the *BIC score* is defined as

$$\mathrm{score}_{BIC}(\mathcal{B} : \mathcal{D}) := l(\hat{\theta}_{\mathcal{B}} : \mathcal{D}) - \frac{\log n}{2}\mathrm{Dim}[\mathcal{B}] \tag{F-4}$$

which can be considered an approximation of the Bayesian score when the data is sufficiently large.

In PGM-Explainer, we use *BIC score* for our structure learning step. One main reason is that *BIC score* is known to be *consistent*. The definition of a score to be *consistent* is given as follows.

Definition 9. *Assume distribution P has a perfect map $\mathcal{B}^*$. We say that a scoring function is consistent if the following properties hold as the amount of data $n \to \infty$, with a probability that approaches* 1*:*

1. *The structure $\mathcal{B}^*$ will maximize the score.*

2. *All structure $\mathcal{B}$ such that $\mathcal{I}(\mathcal{B}) \neq \mathcal{I}(\mathcal{B}^*)$ will have strictly lower score.*

This condition implies that, for all structures containing different set of independence assertions from those in the distribution (or in the perfect map), they will have strictly lower *BIC score*. Hence, by maximizing *BIC score*, PGM-Explainer theoretically searches for the structure containing all and only the dependence assertions encoded in the data.

APPENDIX G
BAYESIAN NETWORK AND DYNAMIC BAYESIAN NETWORK

Bayesian network (BN) [102] is a Probabilistic Graphical model [73], which represents the conditional dependencies among variables via a directed acyclic graph. The edges in a BN provide users with important information, i.e. conditional independence claims, about the relationship between variables in the examined system. BN may be constructed either manually with knowledge of the underlying domain or automatically from datasets by appropriate algorithms. Intuitively, a sparse BN implies its variables can be factorized more easily due to many conditional independence claims in the graph. Because of this intuition, BN has been used to explain predictions made by GNNs [140]. Readers can refer to [73] for a good overview of BNs.

Dynamic Bayesian Network (DBN) [26] can be considered as an extension of BN where temporal information is integrated via edges among variables in adjacent time steps. Denote $\{X^{(t)}\}_{t=0}^{T}$ a set of random vectors with time index t, a DBN is a BN modeling those variables and is specified by:

1. A BN $\mathcal{B}_0$ consists of variables in $X^{(t=0)}$ and the corresponding probability distribution on those variables, i.e. $P_0(X^{(t=0)})$.

2. A set of transition BN $\mathcal{B}^t_{\rightarrow}$ contains variables in $X^{(t)} \cup X^{(t+1)}$. $\mathcal{B}^t_{\rightarrow}$ is equipped with a probability distribution determining the probability of variables in $t+1$ given variables in t, i.e. $P^t_{\rightarrow}(X^{(t+1)}|X^{(t)})$.

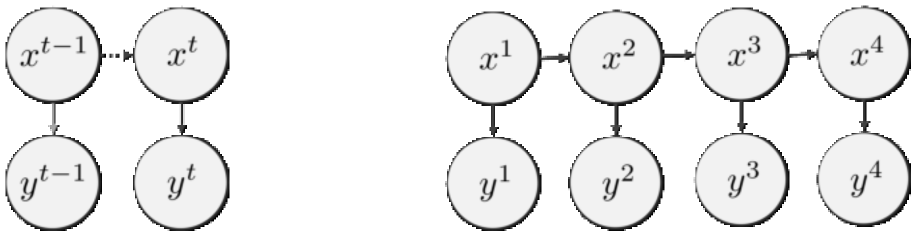
3. The DBN consists of $\mathcal{B}_0$, $\mathcal{B}^t_{\rightarrow}$ and the corresponding probability distributions.

The joint distribution of variables on the DBN then can be factorized by:

$$P\left(X^{(0)},\cdots,X^{(T)}\right) = P_0\left(X^{(0)}\right)\prod_{t=0}^{T-1} P^t_{\rightarrow}\left(X^{(t+1)}|X^{(t)}\right)$$

When the transition probabilities are the same for all time step, i.e. $P^t \rightarrow$ are independent of t, we have one of the most common DBN, which is the Two-Timeslice Bayesian Network

(2TBN) [97]. The 2TBN then can be modeled using only 2 BNs, the first contains the prior probability distribution and the second models the transition, which is the form of all DBNs that we illustrated in our proofs. That reference also shows that any systems with dependency among longer temporal windows, i.e. large timeslice, can be reduced to 2TBNs by adjusting the variables. Figs. G-1a and G-1b show an example 2TBN and its unrolled 4 time-step BN. Readers can find more details of DBNs and how to learn them in [97, 100].



(a) A 2TBN. (b) The unroll BNs.

Figure G-1. Example of the 2TBN and its unroll BN. The intra-slice connections are in solid-lines and the inter-slice connections are in dashed-lines. The brighter solid-line can be omitted as it can be inferred from other edges.

LIST OF REFERENCES

[1] *Captum: A unified and generic model interpretability library for pytorch*, ArXiv **abs/2009.07896** (2020).

[2] G. Alain and Y. Bengio, *Understanding intermediate layers using linear classifier probes*, ICLR (2017).

[3] Raed Alharbi, Minh N. Vu, and My T. Thai, *Evaluating fake news detection models from explainable machine learning perspectives*, ICC 2021 - IEEE International Conference on Communications (2021), 1–6.

[4] ______, *Learning interpretation with explainable knowledge distillation*, 2021 IEEE International Conference on Big Data (Big Data) (2021), 705–714.

[5] Kenza Amara, Rex Ying, Zitao Zhang, Zhihao Han, Yinan Shan, Ulrik Brandes, Sebastian Schemm, and Ce Zhang, *Graphframex: Towards systematic evaluation of explainability methods for graph neural networks*, arXiv preprint arXiv:2206.09677 (2022).

[6] Marco Ancona, Enea Ceolini, Cengiz Öztireli, and Markus Gross, *Towards better understanding of gradient-based attribution methods for deep neural networks*, International Conference on Learning Representations, 2018.

[7] Daniel W. Apley and Jingyu Zhu, *Visualizing the effects of predictor variables in black box supervised learning models*, Journal of the Royal Statistical Society Series B **82** (2020), no. 4, 1059–1086.

[8] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-López, Daniel Molina, Richard Benjamins, et al., *Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai*, Information fusion **58** (2020), 82–115.

[9] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek, *On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation*, PLOS ONE **10** (2015), no. 7, 1–46.

[10] David Baehrens, Timon Schroeter, Stefan Harmeling, Motoaki Kawanabe, Katja Hansen, and Klaus-Robert Müller, *How to explain individual classification decisions*, Journal of Machine Learning Research **11** (2010), no. 61, 1803–1831.

[11] A.L. Barabási and M.Ã. PÃ3sfai, *Network science*, Cambridge University Press, 2016.

[12] D. Bau, J. Y. Zhu, H. Strobelt, A. Lapedriza, B. Zhou, and A. Torralba, *Understanding the role of individual units in a deep neural network*, Proceedings of the National Academy of Sciences (2020).

[13] Kim Been and F Doshi-Velez, *Introduction to interpretable machine learning*, Proceedings of the Tutorial on Interpretable Machine Learning for Computer Vision (Salt Lake City, Utah, United States, 2018), Conference on Computer Vision and Pattern Recognition, 2018.

[14] Yoshua Bengio, Yann Lecun, and Geoffrey Hinton, *Deep learning for ai*, Communications of the ACM **64** (2021), no. 7, 58–65.

[15] John Blitzer, Mark Dredze, and Fernando Pereira, *Biographies, Bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification*, Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics (Prague, Czech Republic), Association for Computational Linguistics, June 2007, pp. 440–447.

[16] Leo Breiman, *Statistical modeling: The two cultures (with comments and a rejoinder by the author)*, Statistical science **16** (2001), no. 3, 199–231.

[17] E. Candes, Y. Fan, L. Janson, and J. Lv, *Panning for gold: Model-x knockoffs for high-dimensional controlled variable selection*, Journal of the Royal Statistical Society: Series B (Statistical Methodology) (2016).

[18] N. Carlini and D. Wagner, *Towards evaluating the robustness of neural networks*, 2017 IEEE Symposium on Security and Privacy (SP), May 2017, pp. 39–57.

[19] Diogo V Carvalho, Eduardo M Pereira, and Jaime S Cardoso, *Machine learning interpretability: A survey on methods and metrics*, Electronics **8** (2019), no. 8, 832.

[20] Joymallya Chakraborty, Kewen Peng, and Tim Menzies, *Making fair ML software using trustworthy explanation*, CoRR **abs/2007.02893** (2020).

[21] Chun-Hao Chang, Elliot Creager, Anna Goldenberg, , and David Duvenaud, *Explaining image classifiers by adaptive dropout and generative in-filling*, arXiv preprint arXiv:1807.08024 (2018).

[22] Frédéric Chazal, David Cohen-Steiner, Leonidas Guibas, Facundo Mémoli, and Steve Oudot, *Gromov-hausdorff stable signatures for shapes using persistence*, Comput. Graph. Forum **28** (2009), 1393–1403.

[23] Charles Corbière, Nicolas THOME, Avner Bar-Hen, Matthieu Cord, and Patrick Pérez, *Addressing failure prediction by learning model confidence*, Advances in Neural Information Processing Systems 32 (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), Curran Associates, Inc., 2019, pp. 2902–2913.

[24] T. M. Cover and J. A. Thomas, *Elements of information theory (wiley series in telecommunications and signal processing)*, USA, 2006.

[25] Ian C. Covert, Scott Lundberg, and Su-In Lee, *Explaining by removing: A unified framework for model explanation*, J. Mach. Learn. Res. **22** (2021), no. 1.

[26] Paul Dagum, Adam Galper, and Eric Horvitz, *Dynamic network models for forecasting*, Uncertainty in Artificial Intelligence (Didier Dubois, Michael P. Wellman, Bruce D'Ambrosio, and Phillipe Smets, eds.), Morgan Kaufmann, 1992, pp. 41–48.

[27] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst, *Convolutional neural networks on graphs with fast localized spectral filtering*, Proceedings of the 30th International Conference on Neural Information Processing Systems, 2016, p. 3844–3852.

[28] B. Dimanov, Umang Bhatt, Mateja Jamnik, and Adrian Weller, *You shouldn't trust me: Learning models which conceal unfairness from multiple explanation methods*, SafeAI@AAAI, 2020.

[29] Gavin Weiguang Ding, Luyu Wang, and Xiaomeng Jin, *AdverTorch v0.1: An adversarial robustness toolbox based on pytorch*, arXiv preprint arXiv:1902.07623 (2019).

[30] Ann-Kathrin Dombrowski, Maximilian Alber, Christopher J. Anders, Marcel Ackermann, Klaus-Robert Müller, and Pan Kessel, *Explanations can be manipulated and geometry is to blame.*, NeurIPS (Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché Buc, Emily B. Fox, and Roman Garnett, eds.), 2019, pp. 13567–13578.

[31] Finale Doshi-Velez and Been Kim, *Towards a rigorous science of interpretable machine learning*, arXiv (2017).

[32] Mengnan Du, Ninghao Liu, and Xia Hu, *Techniques for interpretable machine learning*, Communications of the ACM **63** (2019), no. 1, 68–77.

[33] Alexandre Duval and Fragkiskos Malliaros, *Graphsvx: Shapley value explanations for graph neural networks*, European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD), 2021.

[34] Vijay Prakash Dwivedi, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson, *Benchmarking graph neural networks*, arXiv preprint arXiv:2003.00982 (2020).

[35] H. Edelsbrunner and J. Harer, *Computational topology: An introduction*, Applied Mathematics, American Mathematical Society, 2010.

[36] D. Edwards, *Introduction to graphical modelling*, Springer, New York, USA, 2000.

[37] Andre Esteva, Brett Kuprel, Roberto A. Novoa, Justin Ko, Susan M. Swetter, Helen M. Blau, and Sebastian Thrun, *Dermatologist-level classification of skin cancer with deep neural networks*, Nature **542** (2017), 115.

[38] M. Everingham, L. V. Gool, C. K. I. Williams, J. M. Winn, and A. Zisserman, *The pascal visual object classes (voc) challenge*, International Journal of Computer Vision **88** (2009), 303–338.

[39] Lukas Faber, Amin K. Moghaddam, and Roger Wattenhofer, *When comparing to ground truth is wrong: On evaluating gnn explanation methods*, Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining (2021).

[40] Li Fei-Fei, Rob Fergus, and Pietro Perona, *Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories*, 2004 Conference on Computer Vision and Pattern Recognition Workshop (2004), 178–178.

[41] Aaron Fisher, Cynthia Rudin, and Francesca Dominici, *All models are wrong, but many are useful: Learning a variable's importance by studying an entire class of prediction models simultaneously*, Journal of machine learning research : JMLR **20** (2019), 177.

[42] Jonathan Frankle and Michael Carbin, *The lottery ticket hypothesis: Finding sparse, trainable neural networks*, 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, 2019.

[43] Jerome H. Friedman, *Greedy function approximation: A gradient boosting machine.*, The Annals of Statistics **29** (2001), no. 5, 1189 – 1232.

[44] T. Funke, M. Khosla, M. Rathee, and A. Anand, *Zorro: Valid, sparse, and stable explanations in graph neural networks*, IEEE Transactions on Knowledge &amp; Data Engineering **35** (2023), no. 08, 8687–8698.

[45] Thorben Funke, Megha Khosla, and Avishek Anand, *Zorro: Valid, sparse, and stable explanations in graph neural networks*, arXiv preprint arXiv:2105.08621 (2021).

[46] Amirata Ghorbani, Abubakar Abid, and James Zou, *Interpretation of neural networks is fragile*, Proceedings of the AAAI Conference on Artificial Intelligence **33** (2019), no. 01, 3681–3688.

[47] Robert Ghrist, *Barcodes: The persistent topology of data*, BULLETIN (New Series) OF THE AMERICAN MATHEMATICAL SOCIETY **45** (2008).

[48] R.W. Ghrist, *Elementary applied topology*, CreateSpace Independent Publishing Platform, 2014.

[49] Ian Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep learning*, MIT press, 2016.

[50] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy, *Explaining and harnessing adversarial examples*, International Conference on Learning Representations, 2015.

[51] Mikha . L Gromov, *Groups of polynomial growth and expanding maps*, Publ. Math., Inst. Hautes Étud. Sci (1981), 53–73.

[52] David Gunning and David Aha, *Darpa's explainable artificial intelligence (xai) program*, AI magazine **40** (2019), no. 2, 44–58.

[53] David Gunning, Mark Stefik, Jaesik Choi, Timothy Miller, Simone Stumpf, and Guang-Zhong Yang, *Xai—explainable artificial intelligence*, Science robotics **4** (2019), no. 37, eaay7120.

[54] José Gámez, Juan Mateo, and Jose Puerta, *Learning bayesian networks by hill climbing: Efficient methods based on progressive restriction of the neighborhood*, Data Mining and Knowledge Discovery **22** (2011), 106–148.

[55] Will Hamilton, Zhitao Ying, and Jure Leskovec, *Inductive representation learning on large graphs*, Advances in Neural Information Processing Systems (I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.

[56] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, *Deep residual learning for image recognition*, The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2016.

[57] Wenchong He, Minh N. Vu, Zhe Jiang, and My T. Thai, *An explainer for temporal graph neural networks*, GLOBECOM 2022 - 2022 IEEE Global Communications Conference (2022), 6384–6389.

[58] Juyeon Heo, Sunghwan Joo, and Taesup Moon, *Fooling neural network interpretations via adversarial model manipulation*, NeurIPS, 2019.

[59] G. Hinton and N. Frosst, *Distilling a neural network into a soft decision tree*, 2017.

[60] Hans Hofmann, *UCI machine learning repository*, 1994.

[61] Qiang Huang, Makoto Yamada, Yuan Tian, Dinesh Singh, Dawei Yin, and Yi Chang, *Graphlime: Local interpretable model explanations for graph neural networks*, CoRR **abs/2001.06216** (2020).

[62] Lauren Kirchner Jeff Larson, Surya Mattu and Julia Angwin, *How we analyzed the compas recidivism algorithm*, 2016.

[63] Heinrich Jiang, Been Kim, Melody Y. Guan, and Maya Gupta, *To trust or not to trust a classifier*, Proceedings of the 32nd International Conference on Neural Information Processing Systems (Red Hook, NY, USA), NIPS'18, Curran Associates Inc., 2018, p. 5546–5557.

[64] Zhang Jianming, Lin Zhe, Brandt Jonathan, Shen Xiaohui, and Sclaroff Stan, *Top-down neural attention by excitation backprop*, European Conference on Computer Vision(ECCV), 2016.

[65] Ian T Jolliffe, *Principal component analysis for special types of data*, Springer, 2002.

[66] U. M. Khaire and R. Dhanalakshmi, *Stability of feature selection algorithm: A review*, Journal of King Saud University - Computer and Information Sciences (2019).

[67] Been Kim, Elena Glassman, Briana Johnson, and Julie Shah, *iBCM : Interactive Bayesian case model empowering humans via intuitive interaction*, 2015.

[68] Been Kim, Rajiv Khanna, and Oluwasanmi O Koyejo, *Examples are not enough, learn to criticize! criticism for interpretability*, Advances in neural information processing systems **29** (2016).

[69] Diederik Kingma and Jimmy Ba, *Adam: A method for stochastic optimization*, International Conference on Learning Representations (2014).

[70] ______, *Adam: A method for stochastic optimization*, International Conference on Learning Representations (2014).

[71] Thomas N. Kipf and Max Welling, *Semi-Supervised Classification with Graph Convolutional Networks*, Proceedings of the 5th International Conference on Learning Representations, 2017.

[72] Pang Wei Koh and Percy Liang, *Understanding black-box predictions via influence functions*, International conference on machine learning, PMLR, 2017, pp. 1885–1894.

[73] Daphne Koller and Nir Friedman, *Probabilistic graphical models: Principles and techniques - adaptive computation and machine learning*, The MIT Press, 2009.

[74] Alex Krizhevsky, *Learning multiple layers of features from tiny images*, University of Toronto (2012).

[75] Srijan Kumar, Francesca Spezzano, VS Subrahmanian, and Christos Faloutsos, *Edge weight prediction in weighted signed networks*, Data Mining (ICDM), 2016 IEEE 16th International Conference on, IEEE, 2016, pp. 221–230.

[76] Alexey Kurakin, Ian Goodfellow, and Samy Bengio, *Adversarial examples in the physical world*, ICLR Workshop (2017).

[77] H. Lakkaraju, S. H. Bach, and J. Leskovec, *Interpretable decision sets: A joint framework for description and prediction*, Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, 2016, pp. 1675–1684.

[78] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, Proceedings of the IEEE **86** (1998), no. 11, 2278–2324.

[79] Yann LeCun and Corinna Cortes, *MNIST handwritten digit database*, (2010).

[80] Junhyun Lee, Inyeop Lee, and Jaewoo Kang, *Self-attention graph pooling*, 36th International Conference on Machine Learning, ICML 2019, 2019, pp. 6661–6670.

[81] R. Levie, F. Monti, X. Bresson, and M. M. Bronstein, *Cayleynets: Graph convolutional neural networks with complex rational spectral filters*, IEEE Transactions on Signal Processing **67** (2019), no. 1, 97–109.

[82] Stan Lipovetsky and Michael Conklin, *Analysis of regression in game theory approach*, Applied Stochastic Models in Business and Industry **17** (2001), 319 – 330.

[83] Zachary C. Lipton, *The mythos of model interpretability*, Queue **16** (2018), no. 3, 31–57.

[84] Shuying Liu and Weihong Deng, *Very deep convolutional neural network based image classification using small training sample size*, 2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR) (2015), 730–734.

[85] Scott M. Lundberg, Gabriel Erion, Hugh Chen, Alex DeGrave, Jordan M. Prutkin, Bala Nair, Ronit Katz, Jonathan Himmelfarb, Nisha Bansal, and Su-In Lee, *From local explanations to global understanding with explainable ai for trees*, Nature Machine Intelligence **2** (2020), no. 1, 56–67.

[86] Scott M Lundberg and Su-In Lee, *A unified approach to interpreting model predictions*, Advances in Neural Information Processing Systems 30, 2017, pp. 4765–4774.

[87] Dongsheng Luo, Wei Cheng, Dongkuan Xu, Wenchao Yu, Bo Zong, Haifeng Chen, and Xiang Zhang, *Parameterized explainer for graph neural network*, Advances in Neural Information Processing Systems **33** (2020).

[88] Dimitris Margaritis and Sebastian Thrun, *Bayesian network induction via local neighborhoods*, Advances in Neural Information Processing Systems 12, 2000, pp. 505–511.

[89] Leland McInnes, John Healy, Nathaniel Saul, and Lukas Grossberger, *Umap: Uniform manifold approximation and projection*, The Journal of Open Source Software **3** (2018), no. 29, 861.

[90] ———, *Umap: Uniform manifold approximation and projection*, The Journal of Open Source Software **3** (2018), no. 29, 861.

[91] Shengjie Min, Zhan Gao, Jing Peng, Liang Wang, Ke Qin, and Bo Fang, *Stgsn—a spatial–temporal graph neural network framework for time-evolving social networks*, Knowledge-Based Systems **214** (2021), 106746.

[92] Abduallah Mohamed, Kun Qian, Mohamed Elhoseiny, and Christian Claudel, *Social-stgcnn: A social spatio-temporal graph convolutional neural network for human trajectory prediction*, Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 14424–14432.

[93] C. Molnar, *Interpretable machine learning: A guide for making black box models explainable*, 2023.

[94] Federico Monti, Karl Otness, and Michael M. Bronstein, *Motifnet: A motif-based graph convolutional network for directed graphs*, 2018 IEEE Data Science Workshop (DSW) (2018), 225–228.

[95] W. J. Murdoch, C. Singh, K. Kumbier, Reza Abbasi-Asl, and B. Yu, *Definitions, methods, and applications in interpretable machine learning*, Proceedings of the National Academy of Sciences **116** (2019), no. 44, 22071–22080.

[96] W James Murdoch, Chandan Singh, Karl Kumbier, Reza Abbasi-Asl, and Bin Yu, *Interpretable machine learning: definitions, methods, and applications*, arXiv preprint arXiv:1901.04592 (2019).

[97] Kevin Murphy, *Dynamic bayesian networks: Representation, inference and learning*, Ph.D. thesis, University of California, 01 2002.

[98] Facundo Mémoli and Guillermo Sapiro, *Comparing point clouds.*, SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing, vol. 71, 01 2004, pp. 33–42.

[99] A. M. Nguyen, J. Yosinski, and J. Clune, *Deep neural networks are easily fooled: High confidence predictions for unrecognizable images*, IEEE Conference on Computer Vision and Pattern Recognition (2015).

[100] Roxana Pamfil, Nisara Sriwattanaworachai, Shaan Desai, Philip Pilgerstorfer, Paul Beaumont, Konstantinos Georgatzis, and Bryon Aragam, *Dynotears: Structure learning from time-series data*, ArXiv **abs/2002.00498** (2020).

[101] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala, *Pytorch: An imperative style, high-performance deep learning library*, Advances in Neural Information Processing Systems 32, 2019, pp. 8026–8037.

[102] Judea Pearl, *Chapter 3 - markov and bayesian networks: Two graphical representations of probabilistic knowledge*, Probabilistic Reasoning in Intelligent Systems, Morgan Kaufmann, 1988, pp. 77 – 141.

[103] Judea Pearl and Dana Mackenzie, *The book of why: the new science of cause and effect*, Basic books, 2018.

[104] Judea Pearl and Azaria Paz, *Confounding equivalence in causal inference*, Journal of Causal Inference **2** (2014), no. 1, 75–93.

[105] Nhathai Phan, Minh N. Vu, Yang Liu, Ruoming Jin, Dejing Dou, Xintao Wu, and My T. Thai, *Heterogeneous gaussian mechanism: Preserving differential privacy in deep learning with provable robustness*, International Joint Conference on Artificial Intelligence, 2019.

[106] P. E. Pope, S. Kolouri, M. Rostami, C. E. Martin, and H. Hoffmann, *Explainability methods for graph convolutional neural networks*, 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019, pp. 10764–10773.

[107] Alun Preece, Dan Harborne, Dave Braines, Richard Tomsett, and Supriyo Chakraborty, *Stakeholders in explainable ai*, arXiv preprint arXiv:1810.00184 (2018).

[108] Pranav Rajpurkar, Jeremy A. Irvin, Kaylie Zhu, Brandon Yang, Hershel Mehta, Tony Duan, Daisy Yi Ding, Aarti Bagul, C. Langlotz, Katie S. Shpanskaya, Matthew P. Lungren, and A. Ng, *Chexnet: Radiologist-level pneumonia detection on chest x-rays with deep learning*, ArXiv **abs/1711.05225** (2017).

[109] Jonas Rauber, Wieland Brendel, and Matthias Bethge, *Foolbox v0.8.0: A Python toolbox to benchmark the robustness of machine learning models*, CoRR **abs/1707.04131** (2017).

[110] Michael Redmond, *UCI machine learning repository*, 2011.

[111] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin, *"Why should i trust you?": Explaining the predictions of any classifier*, Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, p. 1135–1144.

[112] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin, *Anchors: High-precision model-agnostic explanations*, AAAI, 2018.

[113] Marko Robnik-Šikonja and Marko Bohanec, *Perturbation-based explanations of prediction models*, Human and Machine Learning: Visible, Explainable, Trustworthy and Transparent (2018), 159–175.

[114] M. Robnik-Šikonja and I. Kononenko, *Explaining classifications for individual instances*, IEEE Transactions on Knowledge and Data Engineering **20** (2008), no. 5, 589–600.

[115] Cynthia Rudin, *Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead*, Nature Machine Intelligence **1** (2019), 206–215.

[116] Sean Saito, Eugene Chua, Nicholas Capel, and Rocco Hu, *Improving lime robustness with smarter locality sampling*, ArXiv **abs/2006.12302** (2020).

[117] W. Samek, A. Binder, G. Montavon, S. Lapuschkin, and K. Müller, *Evaluating the visualization of what a deep neural network has learned*, IEEE Transactions on Neural Networks and Learning Systems **28** (2017), no. 11, 2660–2673.

[118] Benjamin Sanchez-Lengeling, Jennifer Wei, Brian Lee, Emily Reif, Peter Wang, Wesley Qian, Kevin McCloskey, Lucy Colwell, and Alexander Wiltschko, *Evaluating attribution for graph neural networks*, Advances in Neural Information Processing Systems (H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, eds.), vol. 33, Curran Associates, Inc., 2020, pp. 5898–5910.

[119] Mauro Scanagatta, Antonio Salmerón, and Fabio Stella, *A survey on bayesian network structure learning from data*, Progress in Artificial Intelligence 8, 2019, p. 425–439.

[120] Patrick Schwab and Walter Karlen, *CXPlain: Causal Explanations for Model Interpretation under Uncertainty*, Advances in Neural Information Processing Systems (NeurIPS), 2019.

[121] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, *Grad-cam: Visual explanations from deep networks via gradient-based localization*, 2017 IEEE International Conference on Computer Vision (ICCV), Oct 2017, pp. 618–626.

[122] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje, *Learning important features through propagating activation differences*, Proceedings of the 34th International Conference on Machine Learning, vol. 70, 06–11 Aug 2017, pp. 3145–3153.

[123] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman, *Deep inside convolutional networks: Visualising image classification models and saliency maps*, Workshop at International Conference on Learning Representations, 2014.

[124] Karen Simonyan and Andrew Zisserman, *Very deep convolutional networks for large-scale image recognition*, International Conference on Learning Representations, 2015.

[125] Dylan Slack, Sophie Hilgard, Emily Jia, Sameer Singh, and Himabindu Lakkaraju, *Fooling lime and shap: Adversarial attacks on post hoc explanation methods*, AAAI/ACM Conference on Artificial Intelligence, Ethics, and Society (AIES), 2020.

[126] ______, *Fooling lime and shap: Adversarial attacks on post hoc explanation methods*, AAAI/ACM Conference on AI, Ethics, and Society (AIES), 2020.

[127] Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda B. Viégas, and Martin Wattenberg, *Smoothgrad: removing noise by adding noise*, Workshop on Visualization for Deep Learning, ICML (2017).

[128] J.T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, *Striving for simplicity: The all convolutional net*, ICLR (workshop track), 2015.

[129] Erik Štrumbelj and Igor Kononenko, *Explaining prediction models and individual predictions with feature contributions*, Knowledge and Information Systems **41** (2013).

[130] Mukund Sundararajan, Ankur Taly, and Qiqi Yan, *Gradients of counterfactuals*, CoRR (2016).

[131] ______, *Axiomatic attribution for deep networks*, Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017 (Doina Precup and Yee Whye Teh, eds.), Proceedings of Machine Learning Research, vol. 70, PMLR, 2017, pp. 3319–3328.

[132] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, *Rethinking the inception architecture for computer vision*, 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2016, pp. 2818–2826.

[133] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus, *Intriguing properties of neural networks*, International Conference on Learning Representations, 2014.

[134] E Tjoa and C Guan, *A survey on explainable artificial intelligence (xai): Towards medical xai, arxiv*, arXiv preprint arXiv:1907.07374 (2019).

[135] Christopher Tralie, Nathaniel Saul, and Rann Bar-On, *Ripser.py: A lean persistent homology library for python*, The Journal of Open Source Software **3** (2018), no. 29, 925.

[136] Laurens Van der Maaten and Geoffrey Hinton, *Visualizing data using t-sne.*, Journal of machine learning research **9** (2008), no. 11.

[137] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio, *Graph Attention Networks*, International Conference on Learning Representations (2018).

[138] Domen Vres and Marko Robnik-Sikonja, *Better sampling in explanation methods can prevent dieselgate-like deception*, CoRR **abs/2101.11702** (2021).

[139] E. Štrumbelj, I. Kononenko, and M. Robnik Šikonja, *Explaining instance classifications with interactions of subsets of feature values*, Data Knowl. Eng. **68** (2009), no. 10.

[140] Minh Vu and My T. Thai, *Pgm-explainer: Probabilistic graphical model explanations for graph neural networks*, Advances in Neural Information Processing Systems (H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, eds.), vol. 33, Curran Associates, Inc., 2020, pp. 12225–12235.

[141] Minh N. Vu, *Source code for PGM-Explainer*, https://github.com/vunhatminh/PGMExplainer.

[142] Minh N. Vu, Truc D. Nguyen, NhatHai Phan, Ralucca Gera, and My T. Thai, *c-eval: A unified metric to evaluate feature-based explanations via perturbation*, 2021 IEEE International Conference on Big Data, 2021.

[143] Minh N. Vu, Truc D. Nguyen, and My T. Thai, *Neucept: Learn neural networks' mechanism via critical neurons with precision guarantee*, 2022 IEEE International Conference on Data Mining (ICDM), 2022, pp. 488–497.

[144] Sandra Wachter, Brent Mittelstadt, and Chris Russell, *Counterfactual explanations without opening the black box: Automated decisions and the gdpr*, Harv. JL & Tech. **31** (2017).

[145] J. Wang, X. Jing, Z. Yan, Y. Fu, W. Pedrycz, and L. T. Yang, *A survey on trust evaluation based on machine learning*, ACM Computing Surveys (CSUR) **53** (2020).

[146] Minjie Wang, Lingfan Yu, Da Zheng, Quan Gan, Yu Gai, Zihao Ye, Mufei Li, Jinjing Zhou, Qi Huang, Chao Ma, Ziyue Huang, Qipeng Guo, Hao Zhang, Haibin Lin, Junbo Zhao, Jinyang Li, Alexander J Smola, and Zheng Zhang, *Deep graph library: Towards efficient and scalable deep learning on graphs*, ICLR Workshop on Representation Learning on Graphs and Manifolds (2019).

[147] Xiaoyang Wang, Yao Ma, Yiqi Wang, Wei Jin, Xin Wang, Jiliang Tang, Caiyan Jia, and Jian Yu, *Traffic flow prediction via spatial temporal graph neural network*, Proceedings of The Web Conference 2020, 2020, pp. 1082–1092.

[148] Han Xiao, Kashif Rasul, and Roland Vollgraf, *Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms*, ArXiv **abs/1708.07747** (2017).

[149] Zhang Xinyi and Lihui Chen, *Capsule graph neural network*, International Conference on Learning Representations, 2019.

[150] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka, *How powerful are graph neural networks?*, International Conference on Learning Representations, 2019.

[151] C. Yang, A. Rangarajan, and S. Ranka, *Global model interpretation via recursive partitioning*, 2018 IEEE 20th International Conference on High Performance Computing and Communications, 2018, pp. 1563–1570.

[152] Chih-Kuan Yeh, Cheng-Yu Hsieh, Arun Sai Suggala, David I. Inouye, and Pradeep Ravikumar, *On the (in)fidelity and sensitivity of explanations*, NeurIPS, 2019.

[153] Rex Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec, *Source code for gNNExplainer:*, `https://github.com/RexYing/gnn-model-explainer`.

[154] Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec, *Hierarchical graph representation learning with differentiable pooling*, Proceedings of the 32nd International Conference on Neural Information Processing Systems (Red Hook, NY, USA), NIPS'18, Curran Associates Inc., 2018, p. 4805–4815.

[155] Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec, *Gnnexplainer: Generating explanations for graph neural networks*, Advances in Neural Information Processing Systems (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), vol. 32, Curran Associates, Inc., 2019.

[156] Jiaxuan You, Bowen Liu, Rex Ying, Vijay Pande, and Jure Leskovec, *Graph convolutional policy network for goal-directed molecular graph generation*, Proceedings of the 32nd International Conference on Neural Information Processing Systems (Red Hook, NY, USA), NIPS'18, Curran Associates Inc., 2018, p. 6412–6422.

[157] Fuxun Yu, Zhuwei Qin, and Xiang Chen, *Distilling critical paths in convolutional neural networks*, (2018).

[158] John R. Zech, Marcus A. Badgeley, Manway Liu, Anthony B. Costa, Joseph J. Titano, and Eric Karl Oermann, *Variable generalization performance of a deep learning model to detect pneumonia in chest radiographs: A cross-sectional study*, PLOS Medicine **15** (2018), no. 11, 1–17.

[159] Matthew D. Zeiler and Rob Fergus, *Visualizing and understanding convolutional networks*, Computer Vision – ECCV 2014 (Cham) (David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, eds.), Springer International Publishing, 2014, pp. 818–833.

[160] Muhan Zhang and Yixin Chen, *Link prediction based on graph neural networks*, Advances in Neural Information Processing Systems (S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds.), vol. 31, Curran Associates, Inc., 2018.

[161] Yue Zhang, David DeFazio, and Arti Ramesh, *Relex: A model-agnostic relational model explainer*, Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society (2021).

[162] Ling Zhao, Yujiao Song, Chao Zhang, Yu Liu, Pu Wang, Tao Lin, Min Deng, and Haifeng Li, *T-gcn: A temporal graph convolutional network for traffic prediction*, IEEE Transactions on Intelligent Transportation Systems **21** (2019), no. 9, 3848–3858.

[163] Marinka Zitnik, Monica Agrawal, and Jure Leskovec, *Modeling polypharmacy side effects with graph convolutional networks*, Bioinformatics **34** (2018), no. 13, 457–466.

BIOGRAPHICAL SKETCH

Minh N. Vu received his Ph.D. from the University of Florida in 2023 under Dr. My Thai. His research focuses on Explainable Machine Learning. Before that, he received a B.Eng. degree (Hons.) in electronics and telecommunications from the Hanoi University of Science and Technology, Hanoi, Vietnam, and the M.Sc. degree in electrical and computer engineering from The University of Akron, Akron, OH, USA, in 2016 and 2018, respectively.

During his Ph.D. study, Minh has published many papers in top-tier conferences and journals including NeurIPS, IJCAI, BigData IEEE, ICDM, GLOBECOM, and IEEE ICC. The full list of publications that has been accomplished during his PhD is [142, 105, 143, 140, 3, 4, 57].