TOWARDS EXPLAINABLE DEEP MODELS FOR IMAGES, TEXTS, AND GRAPHS

A Dissertation

by

HAO YUAN

Submitted to the Graduate and Professional School of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Chair of Committee,	Shuiwang Ji
Committee Members,	Xia Hu
	Bobak Mortazavi
	Byung-Jun Yoon
Head of Department,	Scott Schaefer

August 2021

Major Subject: Computer Science

ABSTRACT

Deep neural networks have been widely studied and applied to different applications in recent years due to their great performance. Even though deep models are shown to be powerful and promising, most of them are developed as black boxes. However, without meaningful explanations of how and why predictions are made, we do not fully understand their inner working mechanisms. Hence, such models cannot be fully trusted, which prevents their use in critical applications pertaining to fairness, privacy, and safety. This raises the need of explaining deep learning models and investigating several questions; some of those are, what input factors are important to the predictions? how the decisions are made through deep networks? and what is the meaning of hidden neurons? In this dissertation, we investigate different explanation techniques for different types of deep models. In particular, we explore both instance-level and model-level explanations for image models, text models, and graph models.

Understanding deep image models is the most straightforward choice for explaining deep models since images are naturally well presented and can be easily visualized. Hence, we start by proposing a novel discrete masking method for explaining deep image classifiers. Our method follows the generative adversarial network formalism that the deep model to be explained is regarded as the discriminator while we train a generator to explain it. The generator is trained to capture discriminative image regions that should convey the same or similar semantic meaning as the original image from the model's perspective. It produces a probability map from which a discrete mask can be sampled. Then the discriminator is used to measure the quality of the sampled mask and provide feedback for updating the generator. Due to the sampling operations, the generator cannot be trained directly by back-propagation. We propose to update it using the policy gradient. Furthermore, we propose to incorporate gradients as auxiliary information to reduce the search space and facilitate training. We conduct both quantitative and qualitative experiments on the ILSVRC dataset to demonstrate the effectiveness of our proposed method. Experimental results indicate that our method can provide reasonable explanations for both correct and incorrect predictions and

outperform existing approaches. In addition, our method can pass the model randomization test, indicating that it is reasoning the attribution of network predictions.

Unlike image models, text models are more difficult to explain since texts are represented as discrete variables and cannot be directly visualized. In addition, most explanation methods only focus on the input space of the models and ignore the hidden space. Hence, we propose to explain deep models for text analysis by exploring the meaning of hidden space. Specifically, we propose an approach to investigate the meaning of hidden neurons of the convolutional neural network models for sentence classification tasks. We first employ the saliency map technique to identify important spatial locations in the hidden layers. Then we use optimization techniques to approximate the detected information of these hidden locations from input sentences. Furthermore, we develop regularization terms and explore words in vocabulary to explain such detected information. Experimental results demonstrate that our approach can identify meaningful and reasonable explanations for hidden spatial locations. Additionally, we show that our approach can describe the decision procedure of deep text models.

These facts further motivate us to study the explanation techniques for graph neural networks (GNNs). Unlike images and texts, graph data are usually represented as continuous feature matrices and discrete adjacency matrices. The structural information in the adjacency matrices is important, which should be considered when providing explanations. Thus, methods for images and texts cannot be directly applied. Hence, we investigate both instance-level and model-level explanations of GNNs to provide a comprehensive understanding. First, existing methods invariably focus on explaining the importance of graph nodes or edges but ignore the substructures of graphs, which are more intuitive and human-intelligible. To provide instance-level explanations for GNNs, we propose a novel method, known as SubgraphX, to explain GNNs by identifying important subgraphs. Given a trained GNN model and an input graph, our SubgraphX explains its predictions by efficiently exploring different subgraphs with the Monte Carlo tree search. To make the tree search more effective, we propose to use Shapley values as a measure of subgraph importance, which can also capture the interactions among different subgraphs. To expedite computations, we propose

efficient approximation schemes to compute Shapley values for graph data. Our work represents the first attempt to explain GNNs via identifying subgraphs explicitly. Experimental results show that our SubgraphX achieves significantly improved explanations, while keeping computations at a reasonable level. Second, while most existing explanation methods only provide instance-level explanations, none of them can provide high-level understanding. We propose a novel approach, known as XGNN, to explain GNNs at the model-level. Our approach can provide high-level insights and a generic understanding of how GNNs work. In particular, we propose to explain GNNs by training a graph generator so that the generated graph patterns maximize a certain prediction of the model. We formulate the graph generation as a reinforcement learning task, where for each step, the graph generator predicts how to add an edge into the current graph. The graph generator is trained via a policy gradient method based on information from the trained GNNs. In addition, we incorporate several graph rules to encourage the generated graphs to be valid. Experimental results on both synthetic and real-world datasets show that our proposed methods help understand and verify the trained GNNs.

DEDICATION

This dissertation is dedicated to my family who encouraged me to pursue my dreams and finish my dissertation.

ACKNOWLEDGMENTS

First and foremost, I would like to express my sincere gratitude to my advisor, Dr. Shuiwang Ji, whose expertise was invaluable in formulating the research questions and methodology. His invaluable advice, continuous support, and patience have encouraged me in all the time of my Ph.D. study. He is an incredible advisor, an enthusiastic researcher. It is an honor for me to work with him and the experience with him is my lifelong asset. Besides my advisor, I would like to thank the rest of my dissertation committee, Dr. Xia Hu, Dr. Byung-Jun Yoon, and Dr. Bobak Mortazavi, for their brilliant comments and invaluable suggestions.

I also would like to acknowledge all members of the Data Integration, Visualization, and Exploration (DIVE) Laboratory at Texas A&M University for their constructive discussions and valuable collaborations.

Last but not least, I would like to thank my family for all their love, encouragement, and advice. I could not have completed this dissertation without their support.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a dissertation committee consisting of Dr. Shuiwang Ji, Dr. Xia Hu, and Dr. Bobak Mortazavi of the Department of Computer Science & Engineering, and Dr. Byung-Jun Yoon of the Department of Electrical & Computer Engineering.

The experimental studies in Chapter 2 were conducted in part by Dr. Lei Cai. The data analysis in Chapter 3 was conducted in part by Yongjun Chen. The experimental studies in Chapter 4 were conducted in part by Haiyang Yu.

All other work conducted for the dissertation was completed by the student independently.

Funding Sources

My dissertation research work is supported in part by Defense Advanced Research Projects Agency and National Science Foundation grants.

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION

Deep neural networks have been widely applied in many popular areas including computer vision [1, 2, 3, 4], natural language processing [5, 6, 7], and graph data analysis [8, 9]. With the development of convolutional neural networks [10, 4], recurrent neural networks [11], deep attention models [6], and graph neural networks [12], deep models have achieved the state-of-the-art performance in several areas. However, most existing approaches only focus on improving the performance of models and treat deep models as black-boxes. Without reasoning the prediction procedures, we cannot understand deep models, thus making them less trustable. For example, for critical applications such as medical decisions, it is necessary to not only make predictions but also explain how and why the predictions are made. In addition, in several regulations and policies of algorithms, it is required that the data subject has the right to obtain an explanation of the decision reached [13]. These facts raise the need of developing explanation techniques to explain deep neural networks. In this dissertation, we aim at exploring such techniques for different deep models.

Recently, several techniques are proposed to explain deep learning models. Since the study of explanation is still in the early stage, these methods mainly focus on different classification models. In terms of what types of explanations are provided, these methods can be categorized into two lines: model-level [14, 15, 16, 17] and instance-level [18, 19, 20, 21, 22, 23, 24]. The model-level methods provide high-level insights and a generic understanding of how models work. The provided explanations are input-independent and explain the general behaviors of the models. Popular model-level approaches include input optimization [25], dataset searching [14], and input generation [15, 17]. Instead of focusing on high-level explanations, instance-level methods explain the prediction for each input example. These methods generate input-dependent explanations and investigate what features in the input space contribute more to the predictions. Common techniques in this category include gradient-based methods [18, 19], visualizations of intermediate feature maps [20, 21], decomposition-based methods [26, 27, 28], surrogate methods [29, 30], and

perturbation-based methods [22, 23, 24]. These two categories explain deep models from different perspectives and both are reasoning the relationships between the input space and the output space. In contrast, recent studies [31, 32, 33] explore the meaning of the representations in the hidden space.

1.1 Explaining Deep Models for Images, Texts, and Graphs

With the advance of deep learning, deep models are developed for different applications. While deep models are widely applied to different domains, most deep models are focusing on images, text, and graphs. Hence, in this dissertation, we investigate the explainability of deep models for images, texts, and graphs. A straightforward way is to develop general methods that suitable for different data types. However, it is not feasible because these data are naturally different, and their special properties should be considered to provide intuitive and human-intelligible explanations.

First, images are the most natural data representations since they can be easily visualized and intuitively understood. Image data consist of image pixels and contain important locality information. Each pixel is associated with its location information and the pixels with the same numerical values but different locations may have different semantic meanings. In addition, individual pixels are meaningless without considering the neighborhood and the background. For example, a black pixel may belong to the car body, the darkness of the sky, and animal furs. Hence, it is important to consider the relationships among different pixels and encourage continuous explanations when explaining deep image models.

Second, text models are more challenging to understand since texts cannot be visualized and less intuitive. Text data contain word tokens and each word is represented by an embedding vector. Each word in text data has its own semantic meaning, which may be important for the predictions. In addition, text data also contain locality information. Different from images, texts are discrete so image explanation methods may not be suitable for texts. For example, optimization technique [25] can explain image classifiers by providing high-level and abstract images. Such abstract images can be visualized and intuitively understood. However, text data can not be abstracted and explained in the same way.

Last, graph data are special while they widely exist in different real-world applications, such as social networks, chemistry, and biology. Different from images and texts, graph data contain limited locality information that the numbers of node neighbors are not fixed and there is no location information. Graph data are usually represented by node feature matrices and adjacency matrices. The adjacency matrices are discrete and contain important topology and structural information. Hence, methods from image and text domains cannot be directly applied. For example, optimization technique [25] cannot be applied to optimize the adjacency matrices. In addition, graph structures, such as network motifs, are highly related to their functionalities so that the structures should be explicitly considered when providing explanations for graph models.

1.2 Dissertation Outline

In this dissertation, we study the explanation techniques to explain different deep models for images, texts, and graphs. We aim at providing explanations from different views that are human-intelligible and faithful to the models. Our methods focus on both the input space and hidden space, providing both model-level and instance-level explanations.

In Chapter 2, we start our exploration by explaining deep image classifiers since image models are more straightforward to explain compared with text and graph models. Specifically, we develop a perturbation-based method that learns a generator to generate masks to capture important image regions in the input space. Different from existing studies [24, 34] which generate soft masks for explanations, our proposed method provides discrete masks to avoid the "introduced evidence" problem. We first develop a generator that treats the original image as the input and outputs a probability map to indicate the importance of different pixels. Then to obtain the discrete mask, we perform sampling from the probability map. Next, the discrete mask is combined with the original image to obtain a new image retaining important input information. The new image is fed into the image classier to evaluate the quality of the discrete mask and provide guidance for generator training. We employ the policy gradient to enable the back-propagation for the sampling step. Experimental studies show that our method can generate high-quality explanations than existing studies. It is also shown that our generator is reasoning the attribution of network predictions

instead of guessing the foreground and background of input images. Furthermore, the Remove And Retrain evaluation results demonstrate the correctness of our method.

In Chapter 3, we continue our exploration by studying the explainability of deep text models, which is a less explored but challenging topic. Since texts are represented as discrete input tokens, they cannot be directly visualized. Instead of migrating the techniques for image models to the text models, we specifically design a method for text models. Different from most existing works that focus on the input space, we study the meaning of neurons in the hidden layers to build a bridge between the input space and the output space so that our method can explain the whole decision procedure. First, we select the top important spatial locations in the hidden layers by employing the gradient-based saliency map technique. Next, we study what information from the input sentence is detected by each important spatial location by optimizing a randomly initialized input to mimic the original behaviors in these spatial locations. In addition, we develop regularization terms to encourage the optimized input to have similar embeddings for each spatial location. Finally, an overall embedding is obtained to search in the vocabulary and find words to assign meaning to each spatial location. Experimental results show that our method can explain the whole decision procedure layer by layer, starting from the input to the predictions.

In Chapter 4, we further explore the explanation techniques of the deep models for graph data. Compared with image and text domains, the explanation of graph neural networks has not been well noticed. While different instance-level methods are proposed to explain GNNs from different views, they are invariably based on studying important nodes, node features, or edges. However, we argue that subgraph-level explanations are more natural and intuitive to understand. Hence, we propose a novel method, known as SubgraphX, to provide instance-level explanations by identifying important subgraph structures. Specifically, we employ the Monte Carlo Tree Search (MCTS) algorithm as the search algorithm to efficiently explore different subgraphs. Then, since the information aggregations in GNNs can be regarded as the interactions among different structures, we propose to employ Shapley values to measure the importance of different subgraphs. To address the computational limitations of Shapley values, we propose an efficient scheme to efficiently

approximate Shapley values by considering the interactions only within the local neighborhood. Experimental results on both synthetic and real-world datasets demonstrate the effectiveness of our proposed SubgraphX while the computational cost remains reasonable and acceptable.

In Chapter 5, we continue exploring the explainability of deep graph models. Existing methods only focus on providing instance-level explanations and cannot provide high-level insights. Hence, we propose to investigate the model-level explanations for graph models. Our proposed method, XGNN, explains graph models by finding the graph patterns to maximize a certain network behavior, such as a certain class prediction. Input optimization is popular for the model-level explanations of image models but it cannot be applied to graphs. We first propose to explore subgraph structures by graph generation. Then we formulate the graph generation as a reinforcement learning problem that for each step, the generator predicts how to add an edge into the current graph. The generator is trained via the policy gradient to maximize the network behavior when feeding the generated graphs into the graph models. Meanwhile, we incorporate several graph rules to encourage the generated graphs to be valid and human-intelligible. Experimental results on both synthetic data and real-world data show that our proposed method can help understand, verify, and improve graph classification models.

1.3 Contributions

The main contributions of this dissertation can be summarized as below:

- We propose a learning-based method to explain deep image models by generating discrete masks. The mask generation is formulated as a reinforcement learning problem and we train a mask generator to capture important input regions. We also incorporate auxiliary information to reduce the search space. With our discrete masks, the "introduced evidence" problem caused by soft masks can be addressed and the performance is significantly improved.

- We propose an optimization-based method to study the meaning of hidden neurons in deep text classifiers. It first selects important hidden locations using saliency map techniques and then obtains optimized input to mimic the neuron activations for these locations. Sev-

eral regularization terms are incorporated to encourage the explanations to be more human-intelligible. By searching from neighboring words in the vocabulary, our method can explain the semantic meaning of the information detected by hidden layers, thereby explaining the whole decision procedures from input to final predictions.

- We propose a novel method (SubgraphX) to explain deep graph models at the instance-level. Our SubgraphX explores and identifies important subgraphs for the predictions. By incorporating the Monte Carlo tree search algorithm, our method can efficiently examine different subgraphs. To fairly measure their importance, we employ Shapley values and propose an efficient way to approximate Shapley values. The explanations provided by our subgraphs are more intuitive and faithful to the model. The performance is significantly and consistently better than previous GNN explanation methods.

- We propose a novel method (XGNN) to provide model-level explanations for graph neural networks. XGNN can provide high-level explanations for graph models by generating graph patterns that maximize a target prediction. The graph generation is formulated as a reinforcement learning problem and the generator is trained to learn how to add an edge to the existing graph. Meanwhile, we incorporate several graph rules to yield valid and human-intelligible explanations. The general and high-level explanations provided by our XGNN can help us better understand deep graph models.

2. EXPLAINING DEEP IMAGE CLASSIFIERS BY GENERATING DISCRETE MASKS*

In this chapter, we start our exploration by explaining deep image classifiers. To explain the image classifier, we need to study what input image regions are more important. Intuitively, the important input image regions should convey similar semantic meaning as the original images and lead to the same predictions. We proposed to train a generator to predict probability maps and sample discrete masks to capture such discriminative image regions to explain the predictions.

2.1 Introduction

Deep neural networks have shown great performance in several computer vision tasks, such as image classification [35, 36], image segmentation [37, 38, 39], and image generation [40, 41]. Despite their promising results, they are lacking explanations for their predictions, which prevents them from being applied to critical applications. Hence, it is necessary to study the explanation techniques for these models. Recently, several approaches have been proposed to explain deep image models, and they mainly focus on classification models. These methods belong to two main categories: namely feature visualization and saliency maps [31]. First, feature visualization methods explain a model by identifying input patterns that lead to a certain behavior of a neuron or a group of neurons. Such input patterns can be obtained by dataset searching, input optimization, or input generation [14, 15, 16, 17]. The second one, saliency maps, also known as attributions, explain what input pixels or words contribute to the final predictions. Such explanations can be obtained using gradient-based approaches [18, 19], visualizations of intermediate feature maps [20, 21], perturbation-based methods [22, 42, 23], and feature inversion [43, 44]. In addition, these two categories can be combined to investigate the meaning of hidden units. The attribution techniques first select important hidden units, and then feature visualization methods are employed to study the meaning for these hidden neurons [33, 31].

*Reprinted with permission from "Interpreting Image Classifiers by Generating Discrete Masks" by Hao Yuan, Lei Cai, Xia Hu, Jie Wang, and Shuiwang Ji, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Copyright 2020 by IEEE.

In this chapter, we focus on image classification models and explain them using saliency maps. For a given image, its saliency map shares the same dimensions as the image where each pixel of the saliency map indicates the importance of the corresponding pixel in the original image. In this work, we propose to explain deep models by learning discriminative areas and generating saliency maps. Inspired by the generative adversarial networks (GANs) [40, 45], we treat the deep model to be explained as the discriminator and employ a trainable generator to explain it. Given an image, the generator is trained to determine discriminative areas and generate a probability map. Then we sample a discrete mask from the probability map and combine it with the original image to obtain a new image. Next, the new image is fed into the discriminator to evaluate the quality of mask by comparing the new predictions and the original predictions. Intuitively, discriminative areas of the original image should lead to the same or similar prediction as the original predictions. Hence, the generator can be updated using feedbacks from the discriminator. However, since the discrete mask is sampled from the probability map and only contains discrete values, we cannot directly train the generator using back-propagation. We propose to formulate such a problem as a reinforcement learning problem and train the generator using policy gradient. Furthermore, due to the immense search space, we propose to incorporate auxiliary information to reduce the search space and facilitate training. Note that recent work [24] proposes to train a generator to produce soft masks and apply the soft masks to the original images. However, it suffers from the "introduced evidence" problem that soft masks introduce new noise and semantic meaning to images, which affects the quality of explanations. Differently, our method obtains discrete masks by sampling and the "introduced evidence" problem can be largely avoided.

We conduct experiments on ILSVRC dataset to demonstrate the effectiveness of our proposed approach. The visual explanation results show that our method produces high quality explanations which can reasonably explain the model's predictions. Next, quantitative analysis is performed by applying generated saliency maps to weakly supervised object localization task. The localization results indicate that our method outperforms several existing state-of-the-art explanation methods. In addition, our method can pass model randomization test [42] which shows that our method

can captures model behavior and our explanations are depending on model parameters. Finally, evaluations on saliency metric [24] also demonstrates the effectiveness of our proposed approach.

2.2 Related work

Recent survey work [46, 42] has shown that several techniques try to explain deep models using saliency maps. Existing saliency methods can be categorized into two lines. We briefly introduce these methods in this section.

The first line of work obtains saliency maps using the model parameters or features. Gradients are widely employed to indicate the contributions of different input factors. The most straightforward way is to directly compute the derivative of the class score with respect to the input image using the first-order Taylor expansion [18, 47], which can be obtained using back-propagation. Such gradients indicate how much the class score will change if there is a change in each input position, which can be considered as local sensitivity. The Integrated Gradients method [48] proposes to measure global sensitivity by combining different scaled versions of input, which can address the gradient saturation problem. Similarly, recent work [49] can also solve gradient saturation while reducing visual diffusion. It combines the input and gradients by element-wise product to produce saliency maps. In addition, to alleviate noise and visual diffusion in feature maps, SmoothGrad [19] proposes to remove noise from saliency maps by adding different noise to the input and averaging over all saliency maps.

Meanwhile, the hidden feature maps can be employed to produce saliency maps since the spatial information is retained through convolution layers. The CAM technique [21] proposes to combine the feature maps produced by the final convolution layer. However, it can be only applied to CNNs with global average pooling layer right before the final output layer, and the weights between these two layers are used to combine different feature maps. Then Grad-CAM [20] is proposed to address this limitation by combining different hidden feature maps with gradients. Both of them require additional upsampling operations to recover the spatial size so that the obtained saliency maps may not be accurate. In addition, feature inversion methods [50, 44] employ optimization to map the hidden representations back to the input space and study what information in

the input is preserved in hidden representations and what is discarded.

The second line of work is based on input perturbation, which monitors the change of prediction probability for a certain class while occluding different image regions [22, 23, 24]. They treat the networks as the black box and only focus on the input and the output. Even though these methods are not directly studying model parameters or features, they explain the model from another perspective: "output variations w.r.t. input perturbations". Recent work [24] proposes to train a generator to produce soft masks and apply the soft masks to the original images. Then the newly obtained images are expected to lead to target predictions. However, soft masks contain continuous values and may cause the "introduced evidence" problem [24]. Any non-zero and non-one value will set the original pixel to a new value, which may introduce new noise and meaning to the input thus affecting the saliency maps. In addition, existing work [51] shows that compared with soft masks, discrete masks tend to capture target objects more precisely in attention models. Hence, our method proposes to obtain discrete masks by sampling and the "introduced evidence" problem can be largely avoided. In addition, the design of several functions in the prior work [24] is heuristic and requires knowledge about the dataset, then it may not well for biological or medical images. Furthermore, the model in [24] is trained with ground-truth and randomly sampled fake labels, which significantly increase the complexity. As it is not feasible to cover all fake labels, the model can provide good explanations of correct predictions but not for wrong predictions, which is shown in our experiments. Similarly, recent work [34] proposes to sample k important features from inputs and maximize the mutual information between original predictions and predictions of selected features. The Gumbel-Softmax [52] is employed to approximate discrete feature masks and enable the error backpropagation. However, the feature masks are not strictly discrete. Hence this method may still cause the "introduced evidence" problem. In addition, it only samples the top k important features from the input, which may not form a continuous image region.

2.3 Methods

In this section, we propose a novel approach to explain deep learning models by learning discriminative regions and generating explanations. We first present an overview of the proposed

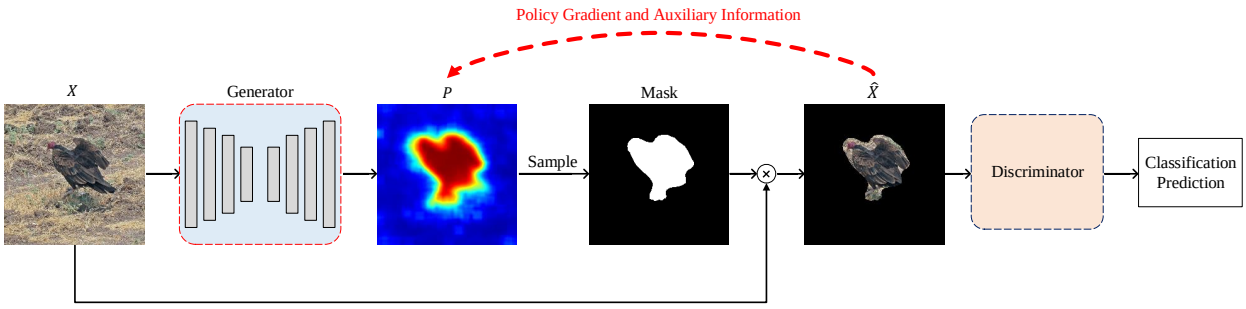

Figure 2.1: Illustrations of the pipeline of our proposed approach. Given an image X, we employ a trainable generator to produce a probability mask P and sample a mask M. Then we perform element-wise product between X and M to obtain a new image $\widehat{X}$. Finally, we feed $\widehat{X}$ to the discriminator to evaluate the quality of P and M.

approach and describe the general pipeline in section 2.3.1. Then the reinforcement learning formulation of our method is discussed in section 2.3.2. Next, we introduce the reward functions for policy training in section 2.3.3. Finally, we propose to incorporate auxiliary information to reduce search space and train the policy in section 2.3.4.

2.3.1 Overview of the Proposed Approach

Traditional GANs consist of two different networks; a generator and a discriminator. These two networks are trained iteratively that the generator learns to capture the data distribution to generate realistic samples while the discriminator is trained to distinguish samples generated from the generator. Inspired by such a mechanism, we propose a novel approach to explain deep neural models in a GAN manner. Specifically, we focus on deep models for image classification. Given a pre-trained model and an input image, our method explains the model by explaining what image regions of the input image are the most discriminative in the model's view. Intuitively, the explanations should capture discriminative image regions for the model's predictions, such that when feeding the explanations of an image to the pre-trained model, it should make the same or similar prediction as the prediction of the original image. We propose a GAN-style architecture for explanations in which there is a generator for generating explanations and a discriminator for evaluating the explanations.

The general pipeline of our proposed method is shown in Figure 2.1. Formally, the pre-trained image classification model serves as the discriminator, denoted as D. The generator is a θ-parameterized generative network, denoted as G_θ. Given an input image $X \in \mathbb{R}^{h \times w}$, the generator G_θ produces a score map P, where each element $p_{i,j}$ is a continous variable and $p_{i,j} \in [0, 1]$. The value of $p_{i,j}$ indicates the probability that the corresponding pixel $x_{i,j}$ in X belongs to important image regions. Such a score map P can be also considered as the saliency map [18] or the attribution map [48]. Then a mask M is sampled from P that each pixel $m_{i,j}$ is a discrete variable whose value can be 0 or 1. The value of $m_{i,j}$ reflects if the corresponding pixel $x_{i,j}$ is selected. Mathematically, it can be written as

$$m_{i,j} \sim \text{Bernoulli}(G_\theta(X)_{i,j}). \tag{2.1}$$

Next, the original image X and the mask M are incorporated together via element-wise product, and output a new image, denoted as $\widehat{X}$. Only important image regions of X are retained in $\widehat{X}$ while the other pixels are set to 0. Then $\widehat{X}$ is fed to the discriminator to output a prediction vector, denoted as $\widehat{y}$ that

$$\widehat{y} = (y_0, \cdots, y_i, \cdots, y_n) = D(X \cdot M), \tag{2.2}$$

where $\cdot$ refers to element-wise multiplication and the y_i is the i^{th} element in $\widehat{y}$ which indicates the probability that $\widehat{X}$ belongs to class i. Note that the discriminator can be any pre-trained image classification model, such as VGG [53], GoogLeNet [54], and ResNet [55]. Hence, our proposed approach can be applied to explain any image classification model. In this work, we choose the VGG network as an example.

In addition, different from traditional GANs, the training of our model is non-adversarial in that only the generator is trained while the discriminator is frozen to provide guidance for improving the generator. Once the generator is well-trained, for any given image, the generated probability map can be used to explain which image regions are important for the model D to make classification predictions. However, there are several challenges to train the generator under such settings. First,

there are sampling operations in our method; then the generator cannot be directly trained through back-propagation. Second, the discriminative image regions should be continuous and only cover related small areas. Without any constraint, the generated probability map may contain many artifacts, and then the sampled pixels in the mask may not be continuous. In this work, we formulate the problem as a reinforcement learning task, train the generator via policy gradient [56, 57], design reward function to measure the quality of sampled masks, and propose to incorporate auxiliary information to reduce the search space and facilitate the training of generator.

2.3.2 A Reinforcement Learning Formulation

As mentioned above, the quality of explanations is discriminated by the pre-trained model D. When feeding the newly generated image $\widehat{X}$ to D, we can obtain the prediction $\widehat{y}$. Then we can calculate the cross-entropy loss that

$$\mathcal{L}_{CE}(\widehat{y}, y) = -\sum_{i=1}^{n} \mathbf{1}\{y = i\} \log \widehat{y}_i, \tag{2.3}$$

where y is a scalar representing the prediction of the original X, n is the number of class, and $\mathbf{1}\{\cdot\}$ denotes the indicator function. Intuitively, training the generator by minimizing this loss can encourage the generator to produce high-quality probability maps where discriminative image regions regarding prediction y yield high probabilities. However, since the mask M is sampled from the probability map P and only contains discrete variables, gradients cannot be back-propagated from the discriminator to the generator. Then the loss $\mathcal{L}_{CE}(\widehat{y}, y)$ cannot be directly used to update the generator. Hence we propose to train the generator in a reinforcement learning manner via policy gradient. The problem can be formulated as a mask generation problem using reinforcement learning.

Formally, we train a generative model G_θ to produce a matrix M based on the input image X. For any element in the matrix M, we have $m_{i,j} \in \{0, 1\}, 1 \leq i \leq h, 1 \leq j \leq w$, where h and w refer to the height and width of X. We assume that the mask is generated row by row ($i = 1$ to $i = h$), and from left to right in each row ($j = 1$ to $j = w$). Then the state $s_{i,j}$ at step (i, j) is

the matrix generated until the previous step. The policy is the generator network G_θ that produces the probability map P. Then the action at step (i, j) is to determine the value of $m_{i,j}$, which can be 0 or 1, based on the policy. In addition, the feedback from the discriminator is employed as the reward. Since the discriminator can only evaluate the whole mask, there is only a final reward for the whole matrix M and no intermediate rewards. Based on [58, 57], the objective of training the generator is maximizing the expected final reward, starting from the start state s_0 with an empty matrix:

$$J(\theta) = \mathbb{E}_{M \sim G_\theta(X|s_0)}[R_f(X, M)], \tag{2.4}$$

where $R_f(X, M)$ is the final reward for the whole mask. Then we can update the parameters θ by policy gradient that

$$\theta = \theta + \alpha \nabla J(\theta), \tag{2.5}$$

where α denotes the learning rate and we can have

$$\begin{aligned} \nabla J \approx & R_f(X, M) \nabla G_\theta(X|s_0) \\ = & R_f(X, M) G_\theta(X|s_0) \log \nabla G_\theta(X|s_0). \end{aligned} \tag{2.6}$$

2.3.3 Reward Function

The reward $R_f(X, M)$ in Equation 2.4 is critical since it provides guidance for updating the generator. Hence, it should correctly reflect the quality of the generated mask M. As the mask is sampled from the probability map P and we employ P to explain the pre-trained discriminator D, the reward function $R_f(X, M)$ can also indicate the quality of explanations. In this section, we carefully design the reward function.

First, given an input image X with its prediction y, the mask M should cover discriminative image regions for class y. Then the newly obtained image $\widehat{X}$ should receive the same prediction as the original prediction y. Hence, the feedback of the discriminator, which is cross entropy loss in

Equation 2.3, can evaluate the quality of the mask M. We use it as the discriminator reward that

$$R_d(X, M) = -\mathcal{L}_{CE}(D(X, M), y). \tag{2.7}$$

Second, the mask M should only cover the most discriminative regions, and ignore background pixels. Hence, we develop an area size reward that the total area size of selected pixels ($m_{i,j} = 1$) should be relatively small, compared with the area size of the original image X. Mathematically, it can be written as

$$R_a(X, M) = -\frac{\sum_{i=1}^{w}\sum_{j=1}^{h} m_{i,j}}{h \times w}. \tag{2.8}$$

In addition, the selected pixels ($m_{i,j} = 1$) in mask M should distribute continuously, which also means fewer artifacts exist in the generated probability map P. Hence, we propose another reward term $R_c(X, M)$ to evaluate the distribution of the selected pixels in M. Formally, it can be written as

$$R_c(X, M) = -\frac{\sum_{i=1}^{w}\sum_{j=1}^{h}(|m_{i,j} - m_{i-1,j}| + |m_{i,j} - m_{i,j-1}|)}{h \times w}. \tag{2.9}$$

Overall, by updating the generator to maximize the discriminator reward $R_d(X, M)$, we encourage the network to capture the discriminative regions in an image. Then the area reward $R_a(X, M)$ encourages our policy to select relatively fewer pixels and only focus on the most discriminative regions. With the continuous $R_c(X, M)$, the selected pixels tend to form continuous regions. We combine these three rewards to serve as the reward function in Equation 2.4, and it can be written as

$$R_f(X, M) = R_d(X, M) + \lambda_1 R_a(X, M) + \lambda_2 R_c(X, M), \tag{2.10}$$

where λ_1 and λ_2 are hyper-parameters for the area reward and the continuous reward.

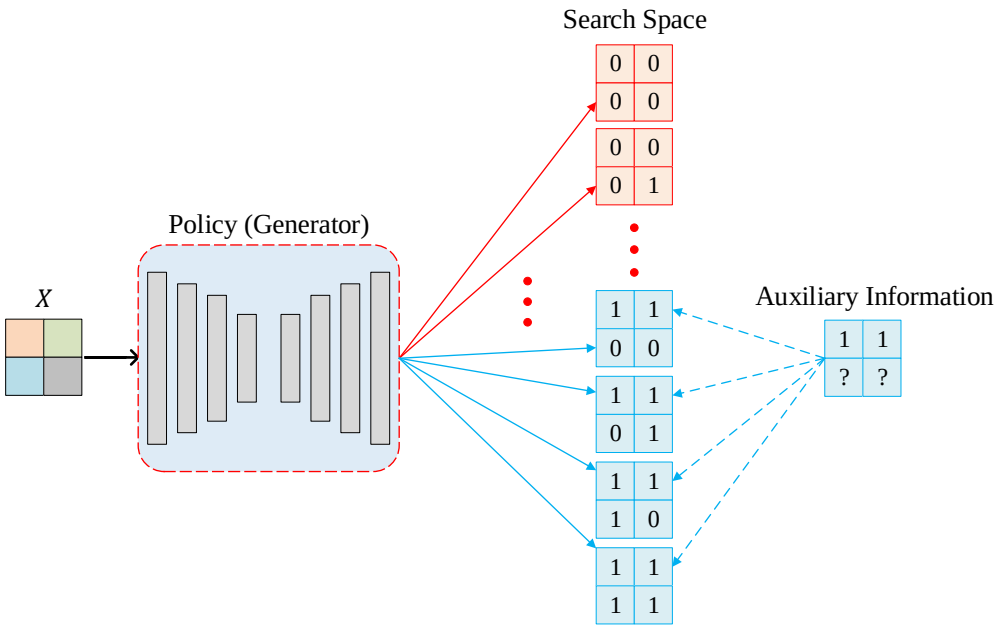

Figure 2.2: Illustrations of how to reduce the search space with auxiliary information. With a 2×2 input X, there are 16 possible paths for the generator to explore, shown as solid lines. With the guidance from auxiliary information (blue dotted lines), the generator is encouraged to search within four states (shown in blue) which share the same labels as the auxiliary information and tends to ignore the masks in red color. Note that question marks mean unknown or unconfident labels. For simplicity, some paths and masks in red are omitted.

2.3.4 Policy Learning with Auxiliary Information

As mentioned above, the generator serves as the policy in our reinforcement learning setting. We first introduce the general structure of our generator network. Given an image X, the generator produces a probability matrix P which shares the same spatial dimensions as the input X. Recently, an encoder-decoder network "U-Net" is proposed for pixel-wise prediction problems and has shown great success in many tasks [37, 38, 39]. Hence, we design our generator as an encoder-decoder network following the general "U-Net" structure, which is shown in Figure 2.1. The encoder consists of several downsampling blocks that each downsampling block employs convolutional layers [10] to extract high-level features from the input image and reduce the spatial size. On the other hand, there are several upsampling blocks in the decoder of our generator to recover the spatial size from high-level features. Each upsampling block contains one transposed convolu-

tional layer [59] and several following convolutional layers. Such a structure can efficiently capture the global relationships between different pixels, and each position of the output depends on all neighbor pixels within its corresponding receptive field. In addition, the spatial information, such as the sizes and locations of different objects and background, can highly impact the explanations. However, since there are multiple downsampling and upsampling operations, such spatial information cannot be perfectly conveyed from the encoder to the decoder. Existing studies [37, 60] have demonstrated that adding skip connections between the encoder and the decoder is beneficial. We employ such an idea and build skip connections between the encoder and the decoder to share spatial information.

Given an image X, the generator produces a probability map P from which we sample a mask M. With the reward $R_f(X, M)$ mentioned above, we can update the generator G_θ. According to [56, 57], the loss function for policy training can be mathematically written as

$$\mathcal{L}_G = -R_f(X, M)\mathcal{L}_{CE}(G_\theta(X), M). \tag{2.11}$$

However, in reinforcement learning, the generator needs to explore the whole search space, which contains a tremendous amount of masks. For example, when the size of images is 224×224, which is a common size used for training ILSVRC dataset [61], each pixel can be selected or not so that the number of possible masks reaches $2^{224\times224}$. Therefore, it is not possible to explore the whole search space and it is challenging to train the policy directly using the loss function in Equation 2.11.

In this work, we propose to employ auxiliary information to reduce the search space and facilitate policy training. The auxiliary information is defined as the guidance information obtained by other methods, and they may not be accurate. As illustrated in Figure 2.2, with an input with size 2×2, there are $2^{2\times2} = 16$ paths leading to 16 possible masks to be explored. With the auxiliary information that positions $(0, 0)$ and $(0, 1)$ are equal to 1 and the remaining positions are unknown, we can guide the generator to search within the masks which share the same labels as the auxiliary

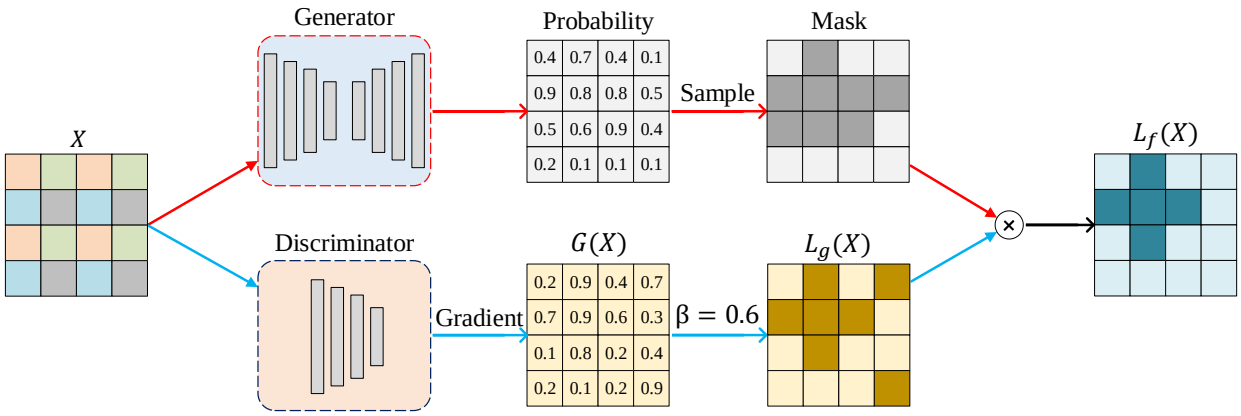

Figure 2.3: An example of incorporating the mask and auxiliary information. Darker colors mean 1 while lighter colors denote 0. We employ the gradients to build our auxiliary information $L_g(X)$ that the obtained gradients are normalized to $[0, 1]$ and thresholded. Then we combine $L_g(X)$ and M by element-wise product to obtain the final labels.

information. Then the generator is encouraged to eliminate 12 possible paths (shown in red) and search within the other 4 masks (shown in blue). Hence, the search space tends to be significantly reduced. Specifically, we employ the gradients to build the auxiliary information in this task, denoted as $L_g(X)$. Even though the gradients can only roughly indicate the importance of different pixels towards the classification, we can observe that the positions with relatively high absolute gradients can capture the discriminative pixels in most cases [18, 22]. Therefore, we calculate the normalized gradients and only use positions with relatively high values as the auxiliary information. Formally, given an input image X, the absolute values of gradients can be approximated by

$$Gr(X) = \left| \frac{\partial S_y}{\partial X} \right|, \tag{2.12}$$

where y denotes the original prediction of X, S_y means the class score of y, and $|\cdot|$ denotes the calculation of absolute values. Then we normalize $Gr(X)$ and obtain the auxiliary information $L_g(X)$ through a threshold β that

$$L_g(i,j) = \begin{cases} 1 & \text{if}(Gr(x_{i,j}) - Gr_{min})/(Gr_{max} - Gr_{min}) \geq \beta \\ 0 & \text{if}(Gr(x_{i,j}) - Gr_{min})/(Gr_{max} - Gr_{min}) < \beta \end{cases}, \tag{2.13}$$

where Gr_{min} indicates the minimum value in $Gr(X)$ and Gr_{max} denotes the maximum value. We present an example of incorporating the auxiliary information and the sampled mask in Figure 2.3. Given an input X, we can obtain a probability map P through the generator and then sample a mask M from P. Meanwhile, we can obtain $Gr(X)$ by computing the gradients of the class score of its original predicted class with respect to input X and then use a threshold to obtain the auxiliary information $L_g(X)$. The positions with deeper colors have values equal to 1 while the light colors mean 0. Then the mask M and the auxiliary information $L_g(X)$ are combined together by element-wise product to produce the final labels $L_f(X)$. In addition, since the reward function $R_d(X, M)$ can only indicate how discriminative the selected pixels are and cannot measure the quality of unselected pixels, we only use the loss obtained from the masked areas to update the policy. Hence, the new loss function can be written as

$$\mathcal{L}_{G_{new}} = -R_f(X, M) \times (M \cdot \mathcal{L}_{CE}(G_\theta(X), M \cdot L_g(X))), \tag{2.14}$$

where $\cdot$ denotes element-wise product. Note that even though we employ the gradients to help update the policy, it is different from directly using gradients as labels. In our proposed method, we only employ gradients to build auxiliary information to reduce search space and the generator is still updated by policy gradient. In addition, the proposed framework can be easily generalized that the discriminator can be any pre-trained model to be explained, the generator can be any pixel-wise prediction model, and the technique to obtain auxiliary information can also be replaced.

2.4 Experimental Studies

In this section, we conduct both qualitative and quantitative evaluations to demonstrate the effectiveness of our proposed method. We first introduce the dataset and our experimental settings in Section 2.4.1 for reproducibility. Then we compare our method with several existing state-of-the-art approaches and report visual explanation results in Section 2.4.2. Next, we evaluate our proposed method through model randomization test in Section 2.4.3. Finally, we present the quantitative analysis based on weakly supervised object localization in Section 2.4.4 and saliency

metric in Section 2.4.5.

2.4.1 Dataset and Experimental Setup

We perform the experiments on the ILSVRC dataset [61], which contains 1000 categories and 1.2 million images for training, and 50,000 images for validation . We follow the VGG network [53] to preprocess images that each image is first resized to $224 \times 224 \times 3$, and then normalized using the mean vector $(123.68, 116.779, 103.939)$. In this work, the model we employed as our discriminator and try to explain is VGG-16. Specifically, we utilize the pre-trained VGG-16 model from Tensorflow Slim Library [62]. The reported top-1 accuracy for ILSVRC validation set is 71.5%, and top-5 accuracy reaches 89.8%. Our discriminator loads such pre-trained model and freeze all parameters during the training of the generator.

Our generator consists of an encoder network and a decoder network. There are three downsampling blocks in our encoder network, and each downsampling block contains three convolutional layers [10]. The stride sizes for the first two convolutional layers are equal to 1 while the final convolutional layer has stride equal to 2 to perform downsampling. The numbers of output channels are doubled for each block, starting from 64. Correspondingly, the decoder network consists of three upsampling blocks that each upsampling block contains one transposed convolutional layer [59] and three convolutional layers. The stride size is set to 2 for the transposed convolutional layer and 1 for each convolutional layer. The number of output channels is halved for the last convolutional layer of the first two blocks, and set to 1 for the final layer in the final block. All other layers retain the same number of output channels as their input. In addition, there is a bottom block connecting the encoder and the decoder, which have two convolutional layers. The stride is set to 1 for both of them, and the first layer doubles the output channels while the second one halves the output channels. In addition, skip connections are performed by simple copying and concatenation. For all layers in our generator, the kernel sizes are set to $(3, 3)$. For all layers except the last layer, batch normalization [63] is applied, and rectified linear unit (ReLU) is employed as activation functions. For the last layer, the sigmoid function is employed as the activation function to generate probabilities between 0 and 1. Note that zero paddings are set to "SAME" for all layers.

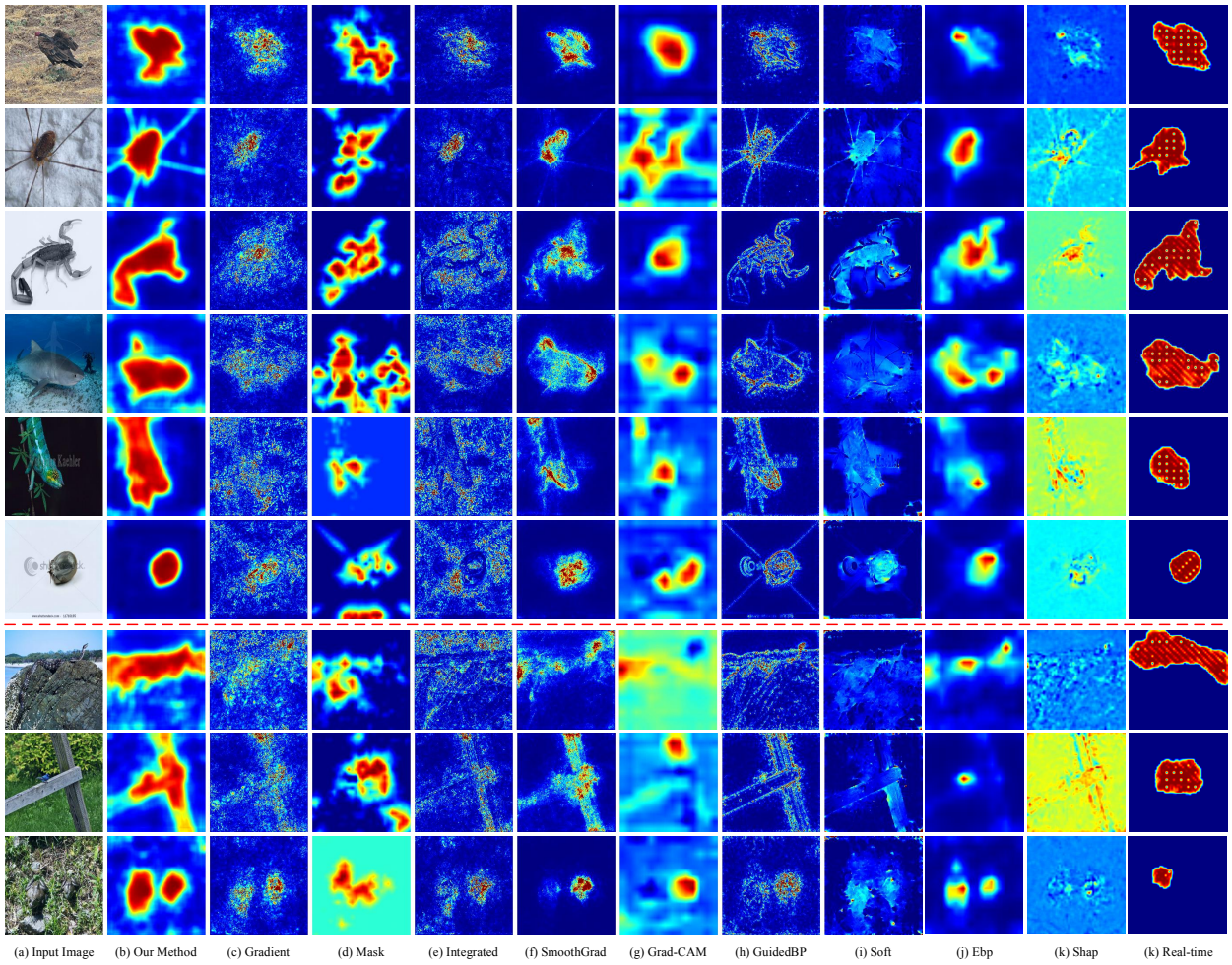

Figure 2.4: Explanation results for the VGG-16 network using different techniques. Different rows show the results for different input images. All saliency maps are normalized to range $[0, 1]$ and visualized using JET colormap. The darker red refers to the higher probability, the green color means the medium probability, and the darker blue means the lower probability. Note that the first six rows are examples with correct predictions while the predictions of the last three rows are wrong.

We implement our method using TensorFlow [64] and conduct our experiments on four Pascal 1080 Ti GPUs. The learning rate for generator is 1×10^{-3} and the batch size is set to 80. The hyper-parameters in Equation 2.10 are set to $\lambda_1 = 3.5$ and $\lambda_2 = 3.0$. The threshold in Equation 2.13 is $\beta = 0.2$. In addition, we apply the Adam optimizer [65] with momentum parameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

2.4.2 Qualitative Evaluations

We conduct experiments to compare the saliency maps generated by different methods qualitatively. These state-of-the-art approaches are Gradient [18], Mask [23], Integrated [48], SmoothGrad [19], Grad-CAM [20], GuidedBP [43], EBP [26], Real-time Saliency [24], and Shapley-value-based method [30]. In addition, to justify the use of reinforcement learning, we train the same generator network with the same loss function but using soft masks, denoted as Soft. We directly apply the probability map P to the original input X and feed $X \cdot P$ to the pretrained model D, which is similar to the existing work [24] and can be trained via standard stochastic gradient descent. For the Shapley-value-based method, we choose to compare with the Shapley GradientExplainer, which combines ideas from Integrated Gradients, SHAP, and SmoothGrad into a single expected value equation. All approaches are visually evaluated using the same VGG-16 model. For each image, we obtain saliency maps from different methods and visualize them using the JET colormap. We report the explanation results of nine different images in Figure 2.4 where each row corresponds to one image. In each row, the leftmost image is the raw image, the second one is the explanation result of our method, and following columns represent results obtained using different approaches in the following order: Gradient, Mask, Intergrated, SmoothGrad, Grad-CAM, GuidedBP, Soft, EBP, Shapley-value-based method, and Real-time Saliency. In each explanation result, the darker red color indicates the higher probability, lighter blue means the lower probability, and green color refers to the medium probability.

In Figure 2.4, the top six rows show the explanations for correctly classified images while the last three rows report the explanations for three misclassified images. The first three rows show the results of three relatively simple examples since these images contain clear backgrounds while the main objects have high-contrast colors compared with their backgrounds. For these three images, our method, SmoothGrad, GuidedBP, Soft, EBP, Shap, and Real-time can provide good explanations to explain the model's predictions. Our method can capture necessary details such as the spider legs and the scorpion tail. The next three rows show three difficult examples in which the main objects are mixed with watermarks. We believe our method provide the most reasonable

explanations which ignore the watermarks while SmoothGrad, GuidedBP and Soft methods fail to do so. From the model's perspective, watermarks can exist in different images so that they should not relate to the predictions. For the seventh row, the model predicts it as "cliff" while its true label is "rock python". Our method, CAM, and Soft can provide reasonable explanations for this wrong prediction that the model focuses more on the rock and reasonably misclassifies it to "cliff". In addition, the prediction of the eighth row is "park bench" while the ground truth is "indigo bunting". Our method and Shap provide good explanations that the model captures the wood whose shape is similar to "park bench". Such explanations indicate that the VGG-16 model may not well capture small objects with low-contrast colors. Next, the last row shows an example whose prediction is "dung beetle" but the label is "bullfrog". Our method explains such a prediction that even though the VGG-16 model captures the objects successfully but misclassifies them to dung beetles. Even though the Real-time saliency method can provide good explanations for correctly classified examples, it cannot provide reasonable explanations for wrong predictions. Overall, we believe our method can successfully explain the predictions of VGG-16 for both correct and incorrect predictions. By comparing different methods, our method tends to provide more reasonable explanations for the predictions. In addition, by comparing our method and Soft, we can show that with reinforcement learning, the explanations are more precise and tend to exclude irrelevant details, such as watermarks.

2.4.3 Model Randomization Test

To demonstrate that our proposed method can capture model behavior instead of generically identifying the foreground of the input images, we evaluate our method using model randomization test [42]. Recent work [42] shows that several widely employed explanation methods are independent to the model parameters, such as GuidedBP [43] and Guided GradCAM [20]. The model randomization test compares the explanations for a trained model, such as the pretrained VGG-16, with the explanations for a randomly initialized model, such as the pretrained VGG-16 with randomly initialized layers. If the explanation method is reasoning the model at hand instead of guessing, the generated explanations should be substantially different for these two cases.

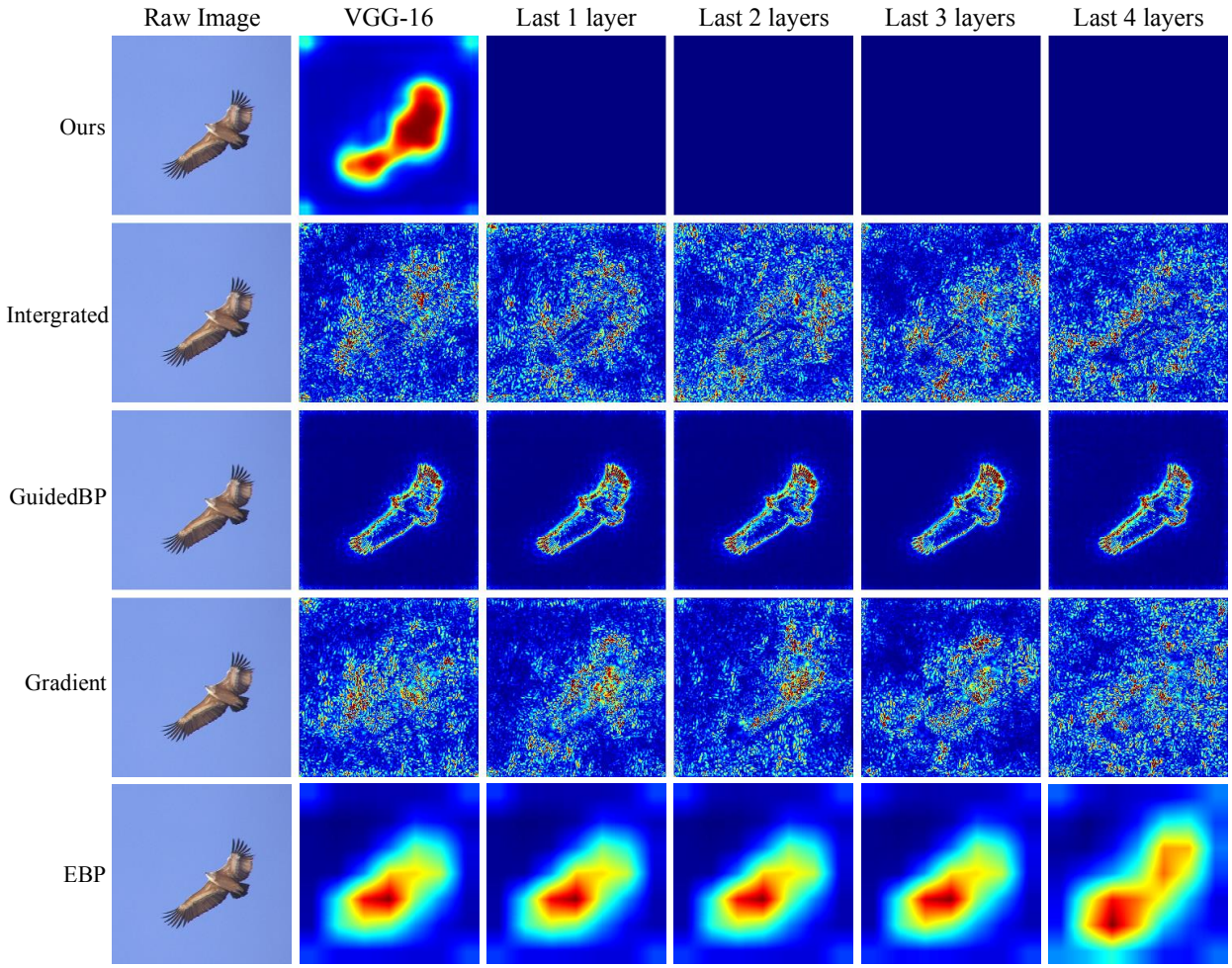

Figure 2.5: The explanation results for model randomization test. In each row, from left to right, we show the raw image, the original explanation, the explanation for VGG-16 with the final layer randomized, the explanation for VGG-16 with the final two layers randomized, the explanation for VGG-16 with the final three layers randomized, and the explanation for VGG-16 with the final four layers randomized.

We compare our proposed method with other baselines via model randomization test in a cascading randomization manner. We first randomize the weights of the final layer in VGG-16 and then provide explanations for this model. Then we evaluate the model with randomized last two layers, the model with randomized last three layers, and the model with randomized last four layers. The results are reported in Figure 2.5, where we compare our method with GuidedBP [43], Intergrated [48], Gradient [18], and EBP [26]. Each row shows the explanations for the same image generated by different methods. In each row, from left to right, we report the raw input, the

original explanation for VGG-16, the explanation for VGG-16 with the final layer randomized, the explanation for VGG-16 with the final two layers randomized, the explanation for VGG-16 with the final three layers randomized, and the explanation for VGG-16 with the final four layers randomized. It is observed that our method provides substantially different explanations for randomized models compared to the original explanation, which means our proposed method can pass the model randomization test. It shows that our method is reasoning the model behavior instead of generically identifying the foreground of the input images. Interestingly, our generated explanations for all randomized models are heatmaps with all zeros. It is reasonable for the following two reasons. First, intuitively, due to the randomized parameters, the models make decisions randomly and the decisions are not related to the input images. Hence, none of the image regions is important to or should contribute to the predictions. Our generated explanations are consistent with such intuition. Second, when training our generator, no matter how the mask is selected, the discriminator reward cannot be maximized while the area reward encourages the area of masks to be as small as possible and finally becomes zero. In addition, we can observe that the Gradient method can also pass the model randomization test. However, the GuidedBP tends to provide almost the same explanations regardless of model randomization, which means it cannot pass the model randomization test. We also observe that the Integrated Gradient method generates highly similar saliency maps for all models. The observations for Gradient, GuidedBP, and Integrated Gradient are consistent with the existing work [42]. For the method EBP [26], the explanations are computed from the feature maps after the last pooling layer, which is between the third-to-last layer and the fourth-to-last layer. It is interesting to observe that if the parameters after the final pooling layer are randomized (the third column to the fifth column in Figure 2.5), the generated explanations of EBP remain the same. Meanwhile, if the parameters before the final pooling layer are randomized (the last column in Figure 2.5), we can observe significant changes in the saliency maps.

Table 2.1: Quantitative comparisions between different approaches using weakly supervised object localization.

Method	**Error Rate**	**Method**	**Error Rate**
Gradient [18]	41.7%	GuidedBP [43]	42.0%
CAM [21]	48.1%	Mask [23]	43.2%
Occlusion [22]	48.6%	Grad-CAM [20]	47.5%
LRP [66]	57.8%	Real-time [24]	39.2%
Soft	47.0%	**Ours**	**37.9%**

2.4.4 Weakly Supervised Object Localization

One popular way to quantitatively measure explanation saliency maps is to apply generated maps to weakly supervised object localization tasks [44, 26, 23]. Even though it cannot precisely measure the explanation quality, the motivation of such evaluation is that reasonable explanations should capture some regions of target objects and hence have overlap with ground truth bounding boxes. We conduct such experiments on ILSVRC validation set which contains 50000 images with humanly annotated bounding boxes. Following the existing work [44, 26, 23] and ILSVRC2014 setting [61], we exclude 1762 images since their bounding box annotations are poor. Note that since the ground truth bounding boxes are annotated for the ground truth label, the targets of explanations are not the original predictions but the ground truth labels. Hence, for such evaluations, our proposed method is trained with ground truth labels instead of predictions. That is, the y in Equation (2.3, 2.7, 2.12) is set as the ground truth label to train the generator.

Given an image X, we can obtain the probability map P. To generate bounding box from P, we first normalize P to be in the range of $[0, 1]$ and then determine which pixels are selected by value thresholding [23] that pixels with values greater or equal to threshold α are selected. Next, the tightest rectangle covering all selected pixels is generated as the final bounding box. Finally, we check if the generated bounding box successfully match the annotated bounding box using intersection over union (IOU) metric. If the IOU score smaller than 0.5, then the generated bounding box is treated as a localization error. Note that if there are multiple annotated bounding

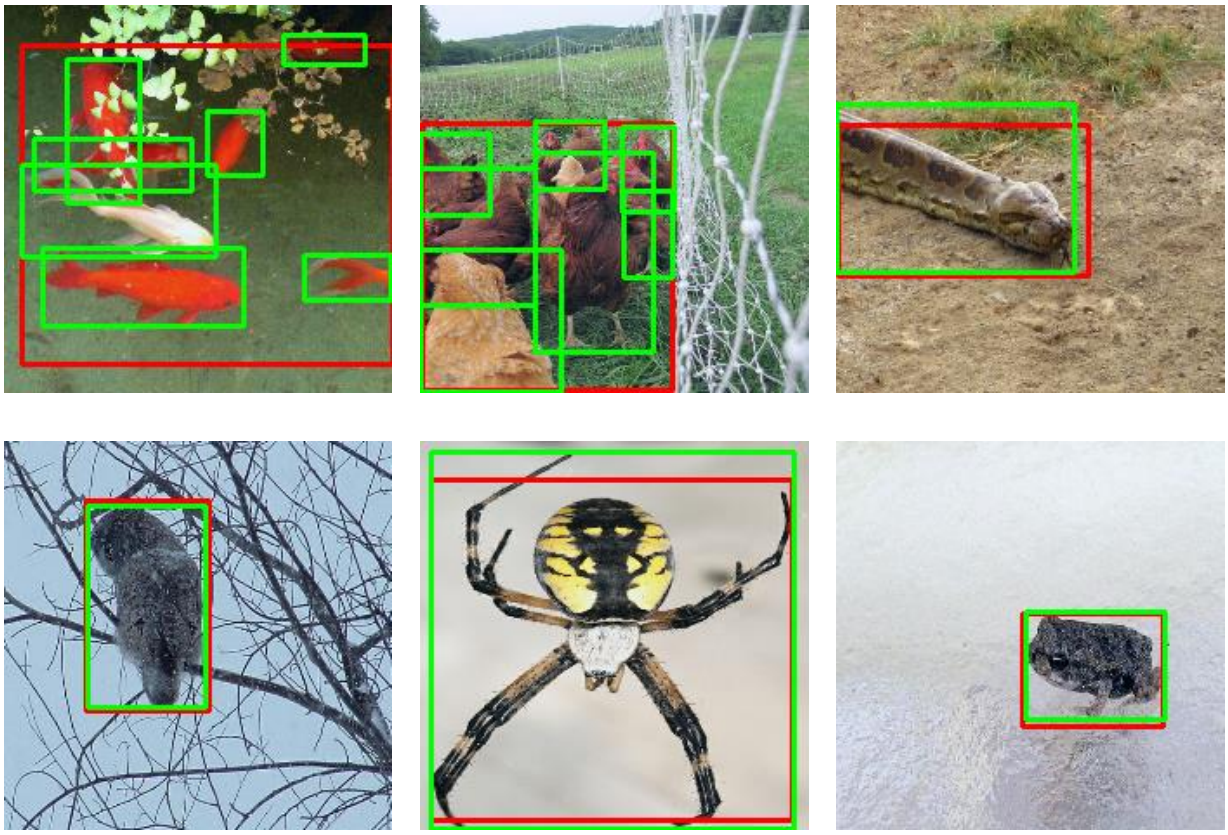

Figure 2.6: The bounding boxes generated by our proposed method for six examples. The red rectangles show the bounding boxes generated by our method and the green ones are annotated ground truth.

boxes in an image, we first find a larger but tightest rectangle covering all annotated bounding boxes and then compute the IOU score based on the larger rectangle and the generated bounding box.

We perform such evaluation for the whole ILSVRC2014 validation set, excluding the poorly annotated images mentioned above. The average localization errors of different approaches are reported in Table 2.1. We compare our method with the several state-of-the-art explanation methods quantitatively. The error rates of these comparing methods are taken from [23]. Our proposed approach achieves the best performance among these eight methods. Note that for our method, the best error rate is achieved when $\alpha = 0.79$. In addition, we compare our method with Soft and Real-time [24]. Clearly, our proposed method outperforms these two methods, which indicates

Table 2.2: Quantitative comparisions between different approaches via saliency metric.

Method	**Score**	**Method**	**Score**
Gradient [18]	0.7999	Mask [22]	0.9059
Integrated [48]	0.7987	Soft	0.8292
Ours	**0.7716**		

discrete masks can better capture target objects. Such observations are consistent with the existing work [51]. In addition, we also report the predicted bounding boxes for six examples in Figure 2.6. The bounding boxes predicted by our method are represented as red rectangles while the manually annotated bounding boxes are shown in green. Our generated bounding boxes can precisely match the annotated ones. Note that for the first and the second images, there are multiple objects and annotated boxes while our predicted bounding boxes can cover all of them. It shows that our proposed method can capture image regions for multiple objects at the same time. Even though such an evaluation cannot directly measure the explanation quality, it reflects that the saliency maps generated by our approach can better highlight the target objects.

2.4.5 Saliency Metric Evaluation

We also evaluate our proposed method by the recently proposed saliency metric [24]. It requires the preserved image regions to yield the same prediction as the original image while the area of the preserved image regions is relatively small. Formally, given an input image X and its original prediction y, we obtain the tightest rectangular box as mentioned in Section 2.4.4. Then we crop the input X based on the rectangular box and resize it to the same size as X, denoted as X_c. Next, we feed X_c to the classifier D and monitor the predicted probability for class y. Then the saliency score is computed as

$$s(a, p) = log(\tilde{a}) - log(\tilde{p}), \tag{2.15}$$

with $\tilde{a} = \max(a, 0.05)$ and $\tilde{p} = \max(p, 0.01)$. Here a means the area of the rectangular as a fraction of the total image size and p is the predicted probability for class y given X_c as the

Table 2.3: The results of the ROAR Evaluation.

	Train Diff		Test Diff	
	$r = 30$	$r = 50$	$r = 30$	$r = 50$
Grad-CAM [20]	-0.018%	-0.108%	-15.36%	-26.98%
Integrated [48]	-0.098%	-0.198%	-0.52%	-2.76%
SHAP [30]	-0.138%	-0.198%	-11.24%	-12.80%
Ours	-0.226%	-0.378%	-18.52%	-23.40%

input. Note that we threshold both a and p to avoid extremely small crops and extremely wrong predictions. We compare our proposed method with Gradient [18], Integrated [48], Mask [22], and Soft. The results are reported in Table 2.2. Our proposed method is shown to achieve a better saliency score than comparing methods. It indicates that the important image regions captured by our method are more discriminative for the model to recognize the original objects. In addition, our method can outperform the Soft, which shows the advantage of employing reinforcement learning for discrete masks.

2.4.6 The Remove and Retrain Evaluation

We further evaluate our proposed method using the recently proposed Remove And Retrain (ROAR) method [67]. It first removes important image pixels for all training and validation images, based on the generated saliency maps, and then retrain the image classification model. Intuitively, if pixels carrying discriminative information are removed, it is more challenging to learn the relations between input images and labels, and hence leads to a significant performance drop. By monitoring how the test accuracy changes, we can understand whether a saliency method captures the discriminative features.

Specifically, we compare our method with SHAP [30], Grad-CAM [20], and Integrated [48] on a 50-classes subset of ILSVRC dataset. We obtain 99.80% training accuracy and 62.92% testing accuracy on this subset with VGG-16. Then we obtain saliency maps for all training and validation images. For each image, based on its saliency map, we select the top $r\%$ of pixels and replace these values with the channel means. Note that r is a pre-defined degradation rate. Finally, we randomly

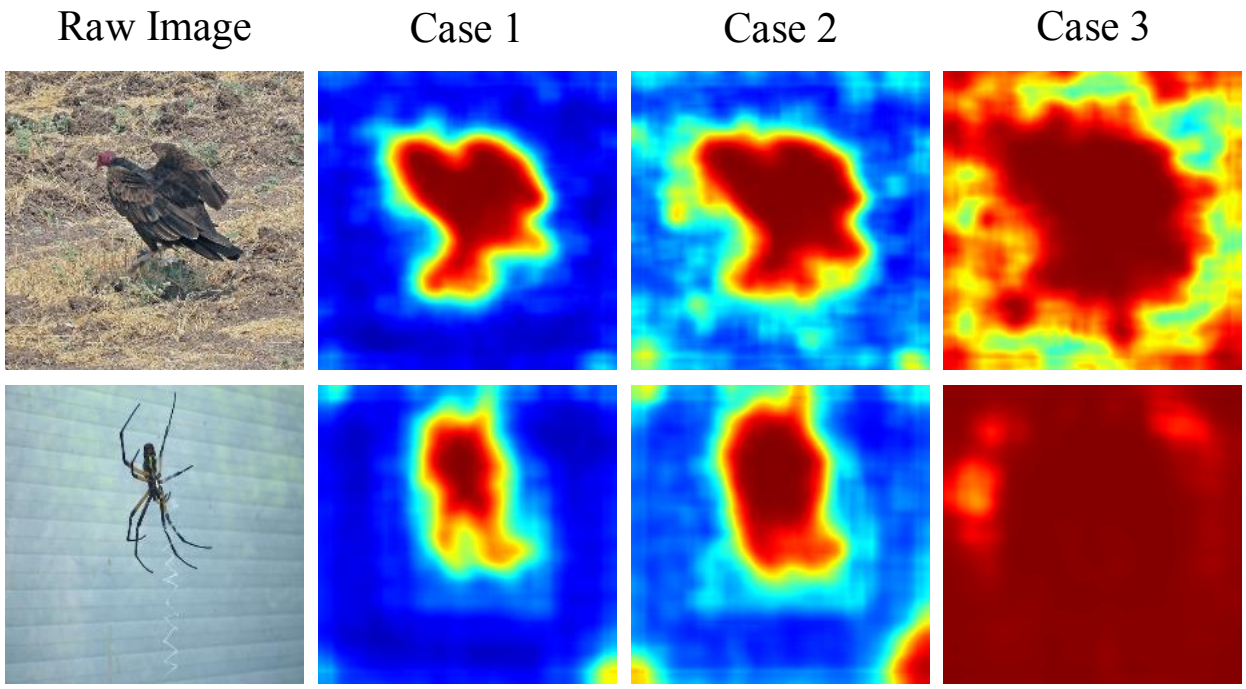

Figure 2.7: The explanation results for the ablation study. In each row, from left to right, we show the raw image, the results of the model with both R_a and R_c reward terms, the results of the model with R_a term but without R_c term, and the results of the model without both terms.

initialize the VGG-16, and re-train it for 50 epochs with modified training data and evaluate it with modified validation data. Note that the learning rate is set to 0.01 and reduced to 0.001 after 30 epochs. The results are reported in Table 2.3. Clearly, for all methods, even when 50% pixels are replaced, the model can still achieve good training performance. The training accuracy only slightly drops compared with the VGG-16 trained with the original subset. It is consistent with the observations in the existing study [67]. However, for our method, Grad-CAM, and SHAP, we can observe significant testing performance degradation. It indicates that the replaced pixels contain important and discriminative information for the model to well capture the data distribution. Note that important pixels identified by our method yield the most test performance degradation when $r = 30$, which further demonstrates the correctness of our method. We believe our method and Grad-CAM outperform the SHAP method in ROAR test since both our method and Grad-CAM capture large continuous image regions while the SHAP method captures several relatively small

discrete regions. Note that the Integrated perform poorly in ROAR test as its saliency maps only contain important discrete pixels.

2.4.7 Ablation Study

In this section, we study the the effectiveness of the area reward R_a in Equation 2.8 and the smoothness reward R_c in Equation 2.9. We consider and compare three different cases; those are, case 1 which is with both R_a and R_c, case 2 which is with R_a but without R_c, case 3 which is without both R_a and R_c. The results are reported in Figure 2.7. We can clearly observe that with both R_a and R_c terms, the model generates the best explanations which only focus on the discriminative areas. Once the R_c term is removed, the generated explanations become non-smooth. In addition, without both R_a and R_c terms, the results are of low quality and tend to cover the whole image. Hence, such results demonstrate that the R_a and R_c terms are useful and necessary.

3. EXPLAINING DEEP TEXT CLASSIFIERS VIA OPTIMIZATION AND REGULARIZATION METHODS*

The effectiveness of our image classifier explanations motivates us to investigate more challenging models. In this chapter, we study the explanation techniques for text models. We propose a novel and intuitive method to investigate the meaning of hidden neurons and explain the whole prediction procedure.

3.1 Introduction

In recent years, deep neural networks have shown great success in many NLP tasks, such as sentence classification [5, 7], natural language generation [57, 68], machine translation [6, 69] and visual question answering [70]. Most existing approaches treat deep neural networks as black-boxes and only focus on the performance. Without understanding the working mechanisms of neural networks, deep models cannot be fully trusted, since we do not know how and why decisions are made. However, due to the complex structures of deep neural networks, it is challenging to explain deep models and their behaviors, especially for NLP tasks that deal with discrete data.

Most existing approaches for explaining NLP models only investigate the relationships between input sentences and output decisions to explore which input words are more important to make decisions [56, 32]. However, the inner workings of networks should also be studied to answer important questions regarding hidden layers, such as which hidden units are more important for a decision and why they are important. To the best of our knowledge, there are no related studies focusing on the explanations of hidden neurons of NLP models.

In this chapter, we propose an approach to explain and understand deep NLP models. Specifically, we focus on convolutional neural networks (CNN) [1] for sentence classification tasks. Our approach employs gradient-based approaches [18] and optimization techniques [14] to se-

*Reprinted with permission from "Interpreting Deep Models for Text Analysis via Optimization and Regularization Methods" by Hao Yuan, Yongjun Chen, Xia Hu, and Shuiwang Ji, *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33(01), 5717-5724, Copyright 2019 by AAAI.

lect spatial locations with high contribution to the decision from hidden layers and study what is detected by these locations. We propose to approximately explain the meaning of detected information using the nearest neighbors of the optimized representation based on the special property of word representations, which imply that words with semantically similar meanings are embedded to nearby points [71]. Experimental results demonstrate that our approach can obtain reasonable and meaningful explanations for hidden units. It is shown that our approach can explain the decision process in NLP models.

3.2 Background and Related Work

Most of the existing explanation approaches are proposed to investigate deep models in computer vision rather than the NLP area. The saliency map techniques study which input pixels are more important to the final decision [18, 46, 44]. The importance of different pixels can be approximated by the gradient of output score with respect to the inputs [22, 43, 72]. The similar idea was applied to NLP models to study which input words contribute more to the prediction [32]. However, such techniques only provide word-level explanation while different words are highly correlated to convey a meaning.

In addition, several approaches focus on feature visualization, which investigates what pattern the hidden neurons of a model try to detect [25, 14, 73, 50, 16]. Optimization techniques are commonly used for such purposes. The key idea is to iteratively update a randomly initialized input to investigate a specific behavior in hidden layers, such as maximizing the activation values of neurons or maximizing the score of a class. The optimized input can then be visualized as abstracted images to reflect the meaning. However, such a technique cannot be directly applied to NLP models since word representations are discrete and the meaning cannot be abstracted. Thus the optimized input is difficult to explain. By combining the above two techniques, [31] investigate the meaning of hidden layers to explain models for image classification tasks. However, as we mentioned above, the optimized input is a sequence of abstract vector representations and cannot be visualized as abstracted texts. We propose an approach to approximately explain the high-level meaning of the optimized input by selecting the neighbors of these vector representations from the

embedding space.

3.3 Methods

As discussed above, it is not enough to only build saliency maps on input sentences to visualize word-level explanation, since different words may combine together to convey a meaning. In addition, without investigating the hidden layers, we still do not understand how the hidden neurons work, and neural networks remain a black box. To better understand deep NLP models, we propose an approach to focus on the contribution and meaning of hidden neurons, thereby allowing us to visually explain the decision process.

3.3.1 Visual Explanations of Hidden Units

In this section, we investigate the explanations of CNN models for sentence classification tasks in NLP. The general structure of CNN models we study is shown in Figure 3.1. Given an input sentence, it first passes through an embedding layer and several convolutional layers. Then it is fed into a max-pooling layer and a fully-connected layer with softmax function to make predictions.

Intuitively, we wish to investigate the hidden units of a deep NLP model so that we can answer three questions; those are, which hidden spatial locations are more important to decisions? what is detected by these spatial locations from input sentences? and what is the meaning of the detected information? However, there are two main challenges for answering these questions; those are, how to explore what is detected by hidden units? and how to explain the detected information? Existing approaches in computer vision cannot be directly applied since word representations are discrete from each other and cannot be abstracted as images.

We first combine the idea of saliency map and optimization to answer the question of what is detected by hidden units. Based on the property of word representations, we propose to approximately explain the meaning of detected information using the nearest neighbors of the optimized representation. Then we develop regularization terms to help explanations. Generally speaking, the explanation procedure consists of three main steps. First, we employ gradient-based approaches to estimate the contributions of different spatial locations in a hidden layer. Based on the magnitude

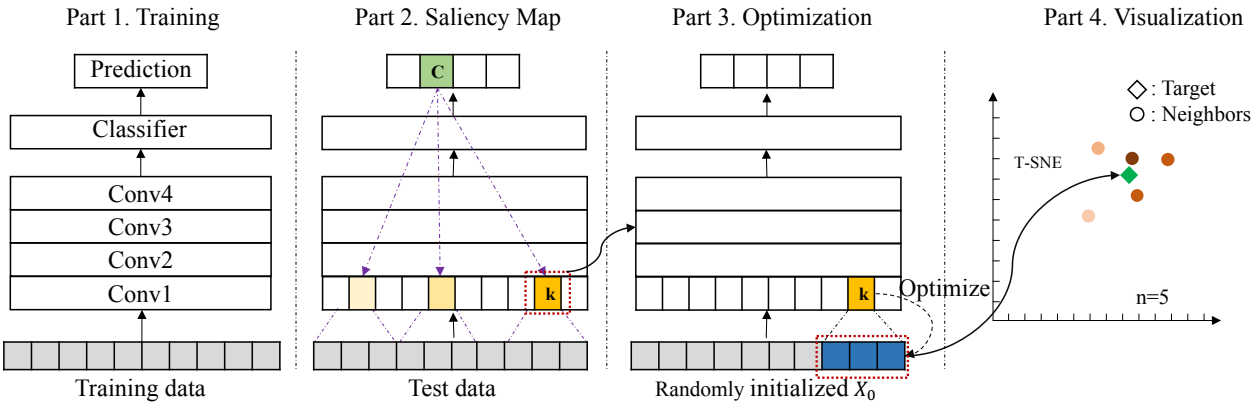

Figure 3.1: Illustration of the overall pipeline of our approach. Part 1 shows the general structure of the CNN model that we try to investigate. After training, we first build saliency maps for different hidden spatial locations, where saliency scores reflect contributions to the final decision. As the example shown in Part 2, the CNN model classifies the test sentence to class c (shown in green). For the conv1 layer, the saliency score is computed for each spatial location, and three spatial locations are selected (highlighted in yellow). Next, for each selected location, optimization technique is employed to determine what is detected from the test sentence. As shown in Part 3, for the spatial location k, a randomly initialized input X_0 is fed to the network and we iteratively update X_0 towards the objective function shown in Equation 3.6. Finally, based on the receptive field of location k (shown in blue with red bounding box), we obtain an overall representation for this receptive field. We search the vocabulary and select word representations with high similarity to the overall representation. Then, the t-SNE is employed to visualize these representations, as shown in Part 4.

of contributions, the spatial locations are sorted, and those with high contribution are selected to be explained in the following steps. Second, to obtain what is detected by different spatial locations in hidden layers, we iteratively update a randomly initialized input via optimization. Finally, the optimized input is a sequence of numerical vectors but such abstract values are hard to explain. Based on the property of word representations that words with semantically similar meanings are embedded to nearby points, we design regularization terms to encourage different vectors in the optimized input to be similar to each other. Then we explore the nearest neighbors [74] in term of cosine similarity to approximately represent the meaning of the target spatial location. The general logic flow of our approach is illustrated in Figure 3.1.

3.3.2 Saliency Maps for Hidden Units

Since there are a large number of neurons in hidden layers, it is not possible to explain each neuron. Hence we employ saliency map techniques to select spatial locations with high contributions for further explanations. The saliency map acts like a heatmap, where saliency scores are estimated by the first order derivatives and reflects the contribution of different neurons. While most of existing approaches build saliency maps to explore the contribution of individual words in input sentences, we study the importance of different hidden spatial locations instead.

Formally, for an input sentence X, the model predicts that it belongs to class c and produces a class score S_c. Let a_{ij} represents the activation vector of the spatial location i of layer j, and its dimension is equal to the number of channels. Also let A_j denotes the activations of layer j, which is a matrix, where each column corresponds to one spatial location. The relationship between the score S_c and A_j is highly non-linear due to the non-linear functions in deep neural networks. Inspired by the strategy in recent work [32, 18], we compute the first-order Taylor expansion as a linear function to approximate the relationship as

$$S_c \approx \mathrm{Tr}(w(A_j)^T A_j) + b, \tag{3.1}$$

where $\mathrm{Tr}(\cdot)$ denotes the trace of a matrix and $w(A_j)$ is the gradient of class score S_c with respect to the layer j. Such gradient can be obtained by using the first order derivative of S_c with respect to the layer A_j as

$$w(A_j) = \frac{\partial S_c}{\partial A_j}. \tag{3.2}$$

For the spatial location i in the layer j, the gradient of S_c with respect to this spatial location is the i^{th} column of $w(A_j)$, denoted as $w(A_j)_i$. Then the saliency score of this location $\mathrm{Score}_c(X)_{i,j}$ is calculated using linear approximation:

$$\mathrm{Score}_c(X)_{i,j} = w(A_j)_i \cdot a_{ij}, \tag{3.3}$$

where $\cdot$ refers to the dot product of vectors.

It is noteworthy that we do not directly use gradients as saliency scores. The reason is that gradients only reflect the sensitivity of the class score when there is a small change in the corresponding spatial location. The employed linear approximation incorporates the activation values to measure how much one spatial location contributes to the final class score. In addition, after training, the weights and parameters in the model are fixed so that the gradient of S_c with respect to a specific spatial location is fixed and does not depend on the input. By using the linear approximation, the saliency score becomes input-dependent.

3.3.3 Input Generation via Optimization

By employing the saliency map technique, we can select spatial locations with high influence on the final decision. However, it is still not clear why they are important. In order to explore this direction, we propose to use optimization techniques to understand what is detected from the input sentence by these spatial locations. The key idea of optimization techniques for explanation is to iteratively update a randomly initialized input towards an objective function. Such optimization procedure is similar to the training of deep neural networks. The main difference is that in such optimization techniques, the parameters of the networks are fixed but the input is optimized. When maximizing the activation value of a certain neuron, the optimized input reflects the pattern that this neuron tries to detect [22, 14]. The activation value of each neuron shows the strength of the pattern detected from inputs. For the neuron k in the spatial location i of hidden layer j, we can obtain an optimized input $\overline{X}_{ijk}$ and the activation value a_{ijk}. When considering spatial locations as a whole, what is detected can be approximated using a weighted sum of $\overline{X}_{ijk}$ and a_{ijk} as

$$\overline{X}_{ij} = \sum_{k=1}^{n} a_{ijk} \overline{X}_{ijk}, \tag{3.4}$$

where n is the number of neurons in the spatial location i of layer j, which is equal to the number of channels.

Such approximations are not efficient since the number of channels can be large, and we need

to obtain an optimized input for each neuron. Furthermore, it is challenging to add regularization since the optimized input is generated for each neuron separately. Hence, we propose to incorporate the activation vector of a spatial location and optimize the input for the whole spatial location. Formally, for a spatial location i of layer j, let a_{ij} represents its activation vector given the input sentence X. We randomly initialize another input X_0 and feed it to the network. For the same spatial location, we obtain another activation vector a'_{ij}. Then we iteratively update the input X_0 towards the following objective function:

$$\max a_{ij} \cdot a'_{ij}, \tag{3.5}$$

where $\cdot$ refers to the dot product of vectors.

3.3.4 Regularization

In Equation 3.5, there is no regularization term for optimization. However, without any regularization, the updating procedure will not converge since the input X_0 can be updated without any constraint, and the target $a_{ij} \cdot a'_{ij}$ keeps increasing. Hence, we add L_2 regularization to the objective function. In addition, in order to explain the optimized input, we propose to add another regularization term, known as the similarity regularization, to make the optimized inputs readily explainable.

Formally, let $\widehat{X_0}$ denotes the receptive field of the spatial location we try to investigate, and l and r are the leftmost and rightmost corresponding indices in $\widehat{X_0}$. Then we have $\widehat{X_0} = [x_{0l}, \cdots, x_{0i}, \cdots, x_{0r}]$, where x_{0i} denotes the i^{th} column of X_0. By adding the regularization terms, the objective function becomes

$$\max a_{ij} \cdot a'_{ij} - \lambda_1 \left\| \widehat{X_0} \right\|_2^2 + \lambda_2 Sim(\widehat{X_0}), \tag{3.6}$$

where $\cdot$ denotes the dot product of vectors, $Sim(\cdot)$ is the similarity term, and λ_1, λ_2 are regularization parameters.

L_2 **Term**: By adding the $L2$ regularization, the optimization procedure converges much faster. Furthermore, the $L2$ term encourages features with high contributions to the target $a_{ij} \cdot a'_{ij}$ to

increase more than others. This is beneficial, since features of high importance can better represent the meaning of hidden spatial locations.

Similarity Term: Intuitively, we try to assign each spatial location an estimated meaning to represent what is detected from the input sentence. After optimization, we obtain multiple vector representations. However, such representations may be very different from each other. In this case, it is challenging to find an overall representation for them. Based on the property of word representation that words with semantically similar meanings localize closer in the embedding space [71, 32], we propose the similarity regularization for optimization, which encourages different vector representations in optimized X_0 to be similar to each other. In this way, these vector representations are encouraged to have similar semantically meanings when mapping back to the word space. Formally, the similarity term is defined as

$$Sim(\widehat{X_0}) = \frac{1}{N} \sum_{\forall i,j} \frac{x_{0i}}{\|x_{0i}\|_2} \cdot \frac{x_{0j}}{\|x_{0j}\|_2}, \tag{3.7}$$

where $\cdot$ refers to dot product of vectors, $N = r - l + 1$ and $i, j \in [l, r]$.

3.3.5 Visualization of Optimized Inputs

By combining saliency maps and optimization, we know which spatial locations in hidden layers contribute most to the final decision. We also obtain an optimized input for each selected hidden spatial location to represent what is detected by this location. However, the optimized input consists of several numerical vectors and is still hard to explain. It is challenging because words representations are discrete so that the optimized representations cannot be mapped to words directly. We propose to find representative words whose vector representations have high cosine similarity with the optimized input as an estimation of the meaning.

Given an optimized input X_0, based on the spatial location we can obtain its receptive field with respect to X_0, denoted as $\widehat{X_0} = [x_{0l}, \cdots, x_{0i}, \cdots, x_{0r}]$. Since we employ the similarity regularization term, different representations x_{0i} are encouraged to be similar. Additionally, in the case of word embedding, similar representations lead to similar semantic meanings. Hence, it is

Dataset	c	N_{train}	N_{test}	$\|V\|$
MR	2	9596	1066	18160
AG's News	4	120000	7600	84252

Table 3.1: The summary statistics of the MR dataset and the AG's News dataset. In the table, c represents the number of classes, N_{train} denotes the number of training examples in the dataset, N_{test} is the number of test examples, and $|V|$ denotes the size of vocabulary.

reasonable to take an average of these representations as an overall approximation as

$$x_{overall} = \frac{1}{N} \sum_{i=l}^{r} x_{0i}. \tag{3.8}$$

It is impossible to find the exact meaning for $x_{overall}$. Instead, we study the neighbors of $x_{overall}$ in the embedding space. We believe the neighbors share similar high-level semantic meaning with $x_{overall}$. Specifically, we compare $x_{overall}$ with different word representations in the vocabulary using cosine similarity and obtain the top words and their corresponding representations. By studying the semantic meaning of these neighbors, we can understand the high-level meaning of the detected information by this spatial location. Finally, these representations can be visualized in the 2D space via dimension reduction techniques, such as t-SNE [75] and principal component analysis [76].

3.4 Experimental Studies

To demonstrate the effectiveness of our approach, we evaluate our methods both quantitatively and qualitatively. We first introduce two datasets we are using and the setup of the experiments in detail. Next, we report the explanation results for several sentence examples. Finally, we present the quantitative evaluations of our methods.

3.4.1 Datasets

We conduct experiments to show the effectiveness of our approach based on two NLP datasets; namely the MR dataset and AG's News dataset. We report the summary statistics of these two datasets in Table 3.1.

	MR	AG's News
Length	56	195
Conv num	3	4
Kernel size	5	5
Conv channel	128, 64, 32	512,256,128,64
Activation	Relu	Relu
Embedding	300	300
Pre-train	Word2vec	Word2vec
Learning rate	2e-4	5e-4
Batch size	128	64

Table 3.2: The CNN models we used for the MR dataset and AG's News dataset. Different columns refer to the network settings for different dataset. *Length*: the length of input sentence; *Conv num*: the number of 1D convolutional layers in the model; *Conv channel*: the number of channels for convolutional layers; *Activation*: activation function in convolutional layers; *Embedding*: dimension of word embedding; *Pre-train*: the type of pre-trained word embedding employed.

MR Dataset: The MR dataset* contains movie review data for sentiment analysis. Each sample in the dataset is a one-sentence movie review and labeled with "positive" or "negative".

AG's News Dataset: The AG's News dataset† is constructed from AG's corpus of news articles. The dataset contains the largest 4 classes of news in the original AG's corpus, where only the title and description are used [7]. The label of each news example depends on the topic of the news, which can be "World", "Sports", "Business" or "Sci/Tech". Each class has 30,000 training examples and 1,900 testing examples.

3.4.2 Experimental Setup

In this section, we introduce the CNN model that we investigate in this work. We then discuss the explanation setup in detail. Finally, we discuss the preprocessing procedure for text inputs.

CNN Model: We build CNN models for both datasets, and the overall structures are shown in Part 1 of Figure 3.1. The input sentence is padded to the same length and fed into the embedding layer, where the word2vec word embedding is employed [71]. Then several 1D convolutional layers [10] and a max-pooling layer are applied. Finally, a fully-connected layer with the softmax

*https://www.cs.cornell.edu/people/pabo/movie-review-data/

†http://www.di.unipi.it/~gulli/AG_corpus_of_news_articles.html

Dataset	MR	AG's News
Our CNN model	79.96%	92.05%
Baseline CNN model	81.50%	91.45%

Table 3.3: Comparison of prediction accuracy between the CNN models we build and the baseline CNNs.

function produces the predictive decision. Detailed descriptions of models are given in Table 3.2.

Explanations: After training, the parameters and vocabulary in CNN models are saved for explanations. These trained parameters in CNN models are reused and fixed during the explanation procedure. Given a test sentence, the saliency map technique returns the top m spatial locations for a hidden layer. We set m equal to 3 in our experiments and focus on the first hidden layer. The input in optimization is randomly initialized using the Xavier initialization method [77]. For the MR dataset, the regularization parameters are set as $\lambda_1 = 0.004$ and $\lambda_2 = 0.02$. For the AG's News dataset, we set $\lambda_1 = 0.002$ and $\lambda_2 = 0.01$. We implement our approach using TensorFlow and conduct our experiments on one Tesla K80 GPU. The learning rate in optimization procedure is set to $2 \times e^{-4}$ and we apply the Adam optimizer [65] with momentum parameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

Preprocessing: The way to preprocess the text data is similar to the existing NLP application [5]. It is noteworthy that we do not convert words to lower case since the meaning of a word is case-sensitive.

3.4.3 Visual Explanation Results

We first report the prediction accuracy of the CNN models that we try to explain. The results are shown in Table 3.3. The CNN models we build can achieve competitive or even better performance compared with the baseline CNNs [5, 7]. The reason why we conduct such comparison is that we wish to show the CNNs we investigate are models with reasonable performance. Next, we present the visual explanation results to demonstrate the effectiveness of our approach.

MR Dataset: For the MR dataset, we show the visualization results for two testing examples; those are, *"As a good old fashioned adventure for kids spirit stallion of the cimarron is a winner"*;

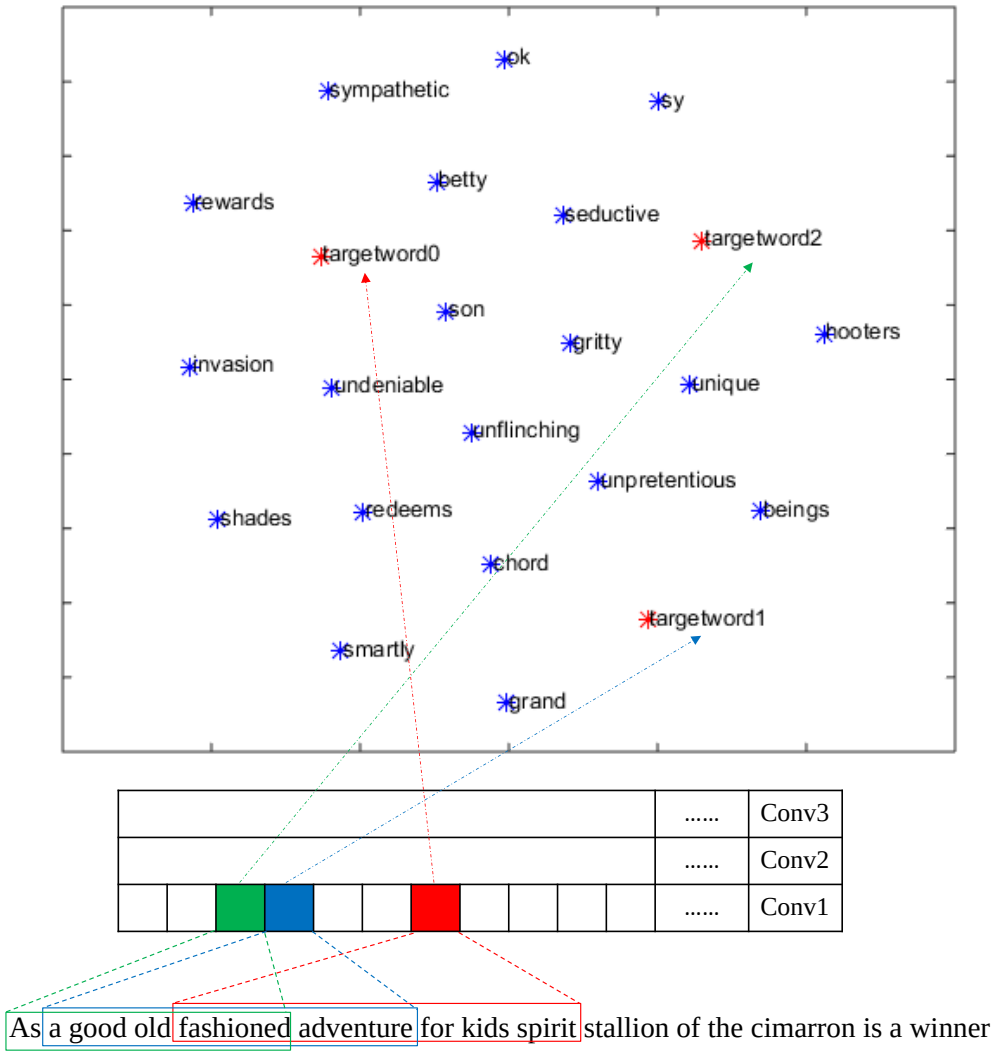

Figure 3.2: The visualization explanation result for the first example for the MR dataset. The middle part of the figure shows the contribution of different spatial locations in hidden layers, where red color means highest contribution to the final decision; blue color refers to the second highest contribution; and green means the third highest contribution. The bounding boxes in different colors correspond to the receptive field of different spatial location. The top part shows the t-SNE visualization of the explanation obtained by our approach. The explanations of target spatial locations are marked as "targetword" and connected to the corresponding spatial locations by dash lines.

and *"Plays like one of those conversations that comic book guy on the simpsons has"*. Clearly, the first example is a positive review while the second one is a negative one. Both of them are correctly classified by the CNN models.

The visual explanation result of the first example is shown in Figure 3.2. As demonstrated,

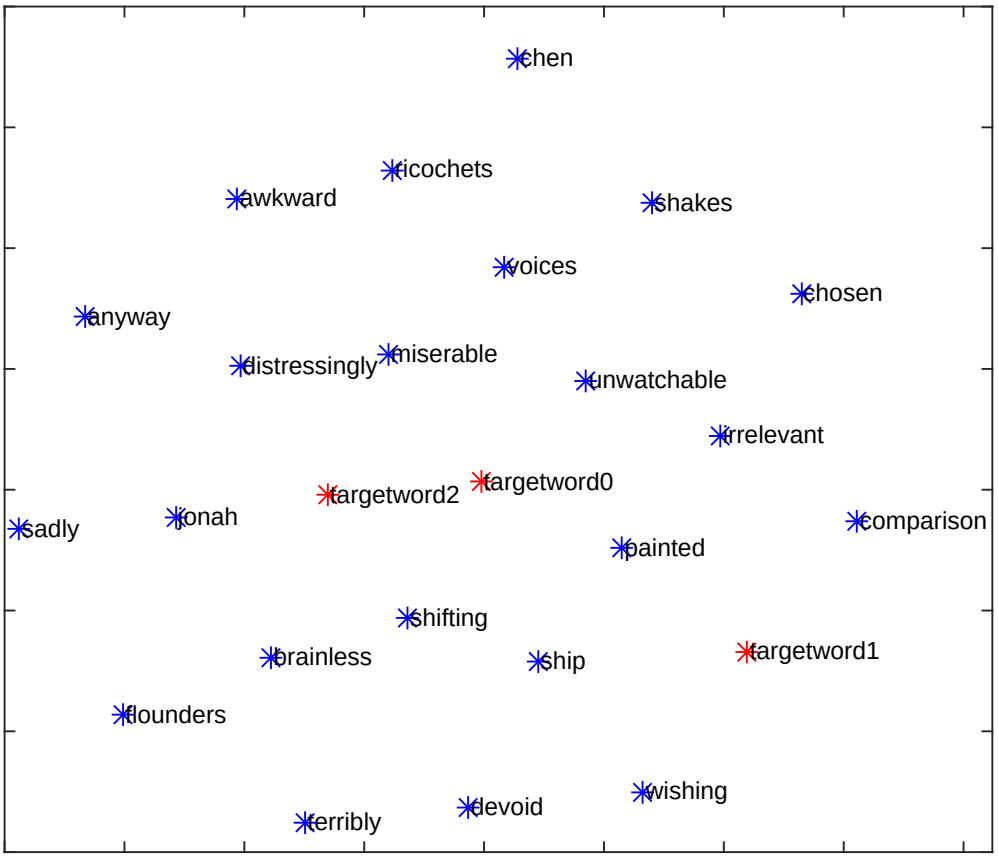

Figure 3.3: The visualization explanation result for the second example for the MR dataset. Only the final result is presented due to space constraints.

three spatial locations (grids in red, blue and green) of the first convolutional layer are selected based on their saliency scores. The bounding boxes reflect the receptive fields of these spatial locations with respect to the input.

The receptive fields contain words like “good”, “fashioned”, “adventure”, and “spirit”, which are commonly used in positive movie reviews. In addition, the top part of Figure 3.2 shows the visual explanation for selected hidden spatial locations. Most of the neighbors identified by our approaches are positive adjectives, such as “unflinching”, “ok”, “smartly”, and “gritty”. We use these neighbors to represent the meanings of hidden spatial locations so that the locations should be explained as positive meaning. This is consistent with their receptive fields and the final positive prediction. Such explanation helps us understand how the decision is made; that is, the information

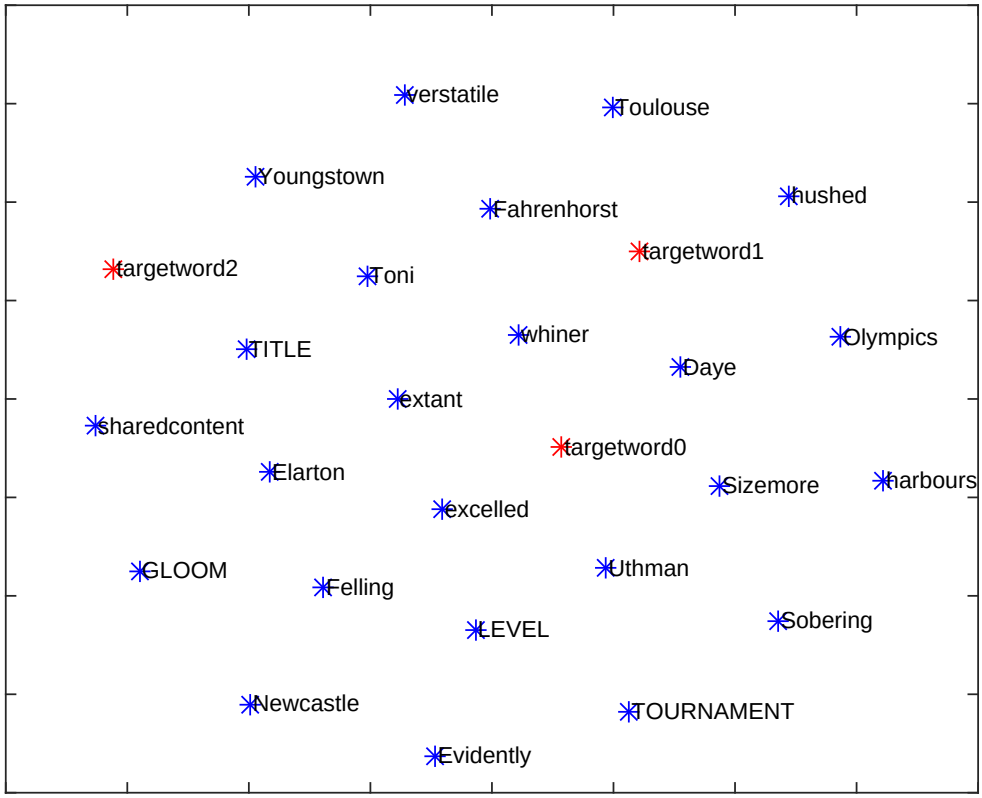

Figure 3.4: The visualization explanation result of the first example for the AG's News dataset.

detected by these spatial locations is positive and these spatial locations have high contribution to the final decision so that the final prediction is positive.

In addition, we show the visualization result of the second MR example in Figure 3.3. Clearly, many words with negative meanings are selected to explain the meaning of hidden spatial locations, such as "terribly", "awkward", "devoid", "unwatchable" and "brainless". Hence, these spatial locations can be explained as negative meaning, and it is consistent with the prediction. We also observe that most of neighbors are adjectives or adverbs.

AG's News Dataset: Similarly, we show the explanation results for two examples from the AG's News dataset. Both of them are correctly classified by CNN models.

The first example with label "sports" is *"Looking at his ridiculously developed upper body, with huge biceps and hardly an ounce of fat, it's easy to see why Ty Law, arguably the best cornerback in football, chooses physical play over finesse. That's not to imply that he's lacking a finesse"*. As

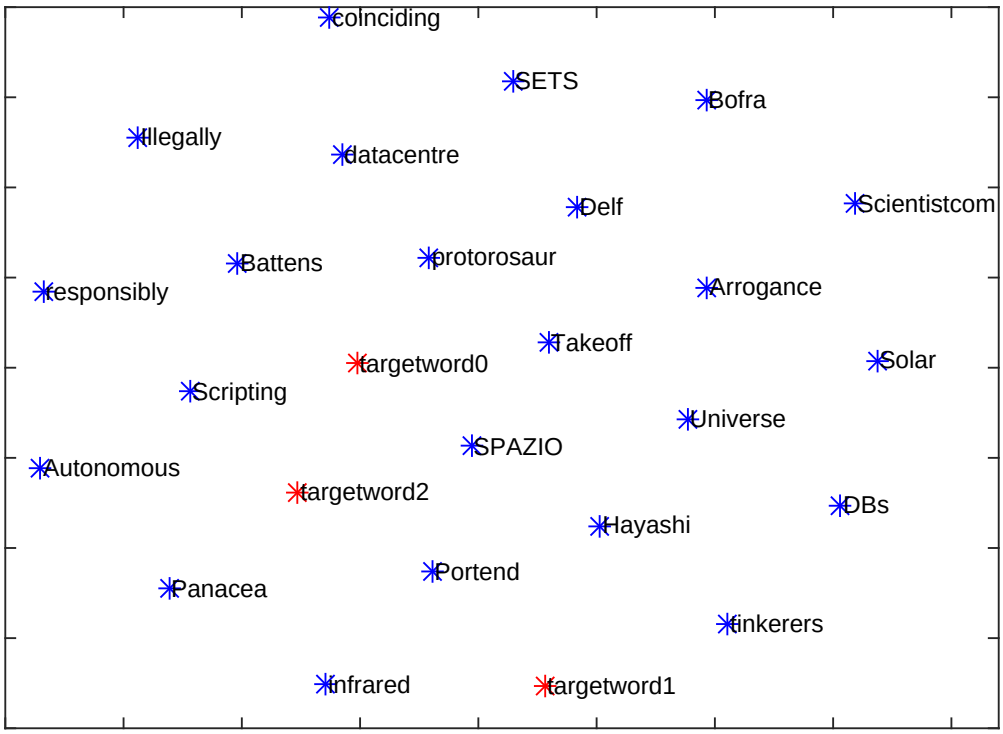

Figure 3.5: The visualization explanation result of the second example for the AG's News dataset.

shown in Figure 3.4, several nouns are selected to explain the hidden locations. Most of them are highly related to the topic "sports", for example, "Toni", "Elarton" and "Fahrenhorst" are names of famous players; "Toulouse " and "Newcastle" are names of famous sports teams. One may argue that such names can be used in many areas and are not limited in topic "sports". We claim that our explanation results are based on the model and datasets where there are only four classes: "World", "Sports", "Business" and "Sci/Tech". When only considering these four types of news, these names are highly related to "sports". Hence, we believe the selected words are reasonable and consistent with the prediction.

The second example is *"Jet Propulsion Lab – Scientists have discovered irregular lumps beneath the icy surface of Jupiter's largest moon, Ganymede"*. Obviously, it belongs to topic "Sci/Tech". The explanation result is shown in Figure 3.5. Similarly, the word selected by our approaches are highly related to "Sci/Tech" topic, such as "Solar", "protorosaur", "datacenter" and "Scientistcom".

Dataset	MR	AG's News
Matching rate	0.934	0.843

Table 3.4: The matching rates for the MR dataset and AG's News dataset.

In addition, it is interesting that for the MR dataset, the explanation results are mostly adjectives and adverbs while the results of AG's News data contains more nouns. This is reasonable since in movie review, the positive or negative meaning is mostly expressed by adjectives and adverbs while the topic of a news is highly related to nouns. Such observation also demonstrates that our approach provides reasonable explanation based on the model and dataset. In conclusion, the words selected by our approach to explain the hidden locations are meaningful and reasonable. They explain the information detected from the input sentence. In addition, such explanations help explain how the decision is made and why the decision is made.

3.4.4 Evaluation of Explainability

Intuitively, if the explanations of hidden spatial locations are meaningful and reasonable, the hidden layers should convey similar high-level meaning compared with the original input sentence. In this section, we explore if the explanations generated by our approach are reasonable. We first introduce how we quantitatively evaluate the explanations. Given an input sentence X, the model classifies it to class c. We obtain the explanations of k locations with the highest contribution to the decision. For each location we use the m nearest neighbors to represent its meaning. In this way, for each input, we have a sentence with km words to explain the hidden layer, denoted as X' . If we feed X' to the same network, we obtain another classification result c'. If c is equal to c', we call it a matching, and it means the explanations of hidden locations shares similar high-level meaning with the input. Here, we focus on the first hidden layer and set $k = 3$ and $m = 5$, .

We conduct such evaluation for the two datasets and the results are reported in Table 3.4. It is obvious that for both datasets, our method provides reasonable explanations for most examples. In addition, our approach has better performance on the MR dataset. We believe the reason is that the length of input examples in the AG's News is much greater than that of MR data. Then it is

more challenging to use the explanations of three locations to represent the meaning of whole input sentences.

4. EXPLAINING DEEP GRAPH CLASSIFIER VIA SUBGRAPH EXPLORATIONS*

We continue our explorations by explaining graph neural networks. Graph data are special since they only have limited locality information but the topology information is important. In this chapter, we propose a novel method, known as SubgraphX, to study instance-level explanations of graph neural networks.

4.1 Introduction

Graph neural networks have drawn significant attention recently due to their promising performance on various graph tasks, including graph classification, node classification, link prediction, and graph generation. Different techniques have been proposed to improve the performance of deep graph models, such as graph convolution [12, 82, 105, 106], graph attention [9, 107], and graph pooling [8, 86, 80]. However, these models are still treated as black boxes, and their predictions lack explanations. Without understanding and reasoning the relationships behind the predictions, these models cannot be understood and fully trusted, which prevents their applications in critical areas. This raises the need of investigating the explainability of deep graph models.

Recently, extensive efforts have been made to study explanation techniques for deep models on images and text [18, 19, 108, 109, 44]. These methods can explain both general network behaviors and input-specific predictions via different strategies. However, the explainability of GNNs is still less explored. Unlike images and texts, graphs are not grid-like data and contain important structural information. Thus, methods for images and texts cannot be applied directly. While several recent studies have developed GNN explanation methods, such as GNNExplainer [110], PGExplainer [111], and PGM-Explainer [112], they invariably focus on explainability at node, edge, or node feature levels. We argue that subgraph-level explanations are more intuitive and useful, since subgraphs can be simple building blocks of complex graphs and are highly related to

*Part of the data reported in this chapter is reprinted with permission from "On Explainability of Graph Neural Networks via Subgraph Explorations" by Hao Yuan, Haiyang Yu, Jie Wang, Kang Li, and Shuiwang Ji, *Proceedings of the 38th International Conference on Machine Learning*, accepted, pending publication, Copyright 2021 by PMLR.

the functionalities of graphs [93, 91].

In this work, we propose the SubgraphX, a novel GNN explanation method that can identify important subgraphs to explain GNN predictions. Specifically, we propose to employ the Monte Carlo tree search algorithm [113] to efficiently explore different subgraphs for a given input graph. Since the information aggregation procedures in GNNs can be interpreted as interactions among different graph structures, we propose to employ Shapley values [114] to measure the importance of subgraphs by capturing such interactions. Furthermore, we propose efficient approximation schemes to Shapley values by considering interactions only within the information aggregation range. Altogether, our work represents the first attempt to explain GNNs via identifying subgraphs explicitly. We conduct both qualitative and quantitative experiments to evaluate the effectiveness and efficiency of our SubgraphX. Experimental results show that our proposed SubgraphX can provide better explanations for a variety of GNN models. In addition, our method has a reasonable computational cost given its superior performance.

4.2 Related Work

4.2.1 Graph Neural Networks

Graph neural networks have demonstrated their effectiveness on different graph tasks. Several approaches are proposed to learn representations for nodes and graphs, such as GCNs [12], GATs [9], and GINs [79], etc. These methods generally follow an information aggregation scheme that the features of a target node are obtained by aggregating and combining the features from its neighboring nodes. Here we use GCNs as an example to illustrate such information aggregation procedures. Formally, a graph $\mathcal{G}$ with m nodes can be represented by an adjacency matrix $A \in \{0,1\}^{m\times m}$ and a feature matrix $X \in \mathbb{R}^{m\times d}$ assuming that each node is associated with a d-dimensional feature vector. Then the aggregation operation in GCNs can be mathematically written as $X_{i+1} = \sigma(D^{-\frac{1}{2}}\hat{A}D^{-\frac{1}{2}}X_iW_i)$, where X_i denotes the output feature matrix of $i-$th GCN layer and X_0 is set to $X_0 = X$. The node features are transformed from $X_i \in \mathbb{R}^{m\times c_i}$ to $X_{i+1} \in \mathbb{R}^{m\times c_{i+1}}$. Note that $\hat{A} = A + I$ is employed to add self-loops and D is a diagonal node degree matrix to

perform normalization on $\hat{A}$. In addition, $W_i \in \mathbb{R}^{c_i \times c_{i+1}}$ is a learnable weight matrix to perform linear transformations on features and $\sigma(\cdot)$ is the non-linear activation function.

4.2.2 Explainability in Graph Neural Networks

Even though explaining GNNs is crucial to understand and trust deep graph models, the explainability of GNNs is still less studied, compared with the image and text domains. Recently, several methods are proposed specifically to explain deep graph models. These methods mainly focus on explaining GNNs by identifying important nodes, edges, node features. However, none of them can provide input-dependent subgraph-level explanations, which is important for understanding graph models. In this section, we briefly discuss several existing methods for explaining GNNs at instance-level.

- **SA** [89] directly employs the squared values of gradients as the importance scores of different input features. It can be directly computed by back-propagation, which is the same as network training but the target is input features instead of model parameters. Note that the input features can be graph nodes, edges, or node features. It assumes that higher absolute gradient values indicate the corresponding input features are more important. While it is simple and efficient, it has several limitations. First, SA can only reflect the sensitivity between input and output, which cannot accurately show the importance. In addition, it also suffers from saturation problems [49]. In the saturation regions of the model, where the model output changes minimally with respect to any input change, the gradients can hardly reflect the contributions of inputs.

- **Guided BP** [89] shares a similar idea with SA but modifies the procedure of backs propagating gradients. Since negative gradients are challenging to explain, Guided BP only back propagates positive gradients while clipping negative gradients to zeros. Then only positive gradients are used to measure the importance of different input features. Note that Guided BP shares the same limitations as SA.

- **CAM** [115] maps the node features in the final layer to the input space to identify impor-

tant nodes. It requires the GNN model to employ a global average pooling (GAP) layer and a fully-connected (FC) layer as the final classifier. Specifically, CAM takes the final node embeddings and combines different feature maps by weighted summations to obtain importance scores for input nodes. Note that the weights are obtained from the final fully-connected (FC) layer connected with the target prediction. This approach is also very simple and efficient but still has several major limitations. First, CAM has special requirements for the GNN structure, which limits its application and generalization. Second, it assumes that the final node embeddings can reflect the input importance, which is heuristic and may not be true. Furthermore, it can only explain graph classification models and cannot be applied to node classification tasks.

- **Grad-CAM** [115] extends the CAM to general graph classification models by removing the constraint of the GAP layer. Similarly, it also maps the final node embeddings to the input space to measure node importance. However, instead of using the weights between the GAP output and FC output, it employs gradients as the weights to combine different feature maps. Specifically, it first computes the gradients of the target prediction with respect to the final node embeddings. Then it averages such gradients to obtain the weight for each feature map. Compared with the CAM, Grad-CAM does not require the GNN model to employ a GAP layer before the final FC layer. However, it is also based on heuristic assumptions and cannot explain node classification models.

- **GNNExplainer** [110] learns soft masks for edges and node features to explain the predictions via mask optimization. The soft masks are randomly initialized and treated as trainable variables. Then GNNExplainer combines the masks with the original graph via element-wise multiplications. Next, the masks are optimized by maximizing the mutual information between the predictions of the original graph and the predictions of the newly obtained graph. Even though different regularization terms, such as element-wise entropy, are employed to encourage optimized masks to be discrete, the obtained masks are still soft masks so that it

cannot avoid the "introduced evidence" problem. In addition, the masks are optimized for each input graph individually and hence the explanations may lack a global view.

- **PGExplainer** [111] learns approximated discrete masks for edges to explain the predictions. It trains a parameterized mask predictor to predict edge masks. Given an input graph, it first obtains the embeddings for each edge by concatenating node embeddings. Then the predictor uses the edge embeddings to predict the probability of each edge being selected, which can be treated as the importance score. Next, the approximated discrete masks are sampled via the reparameterization trick. Finally, the mask predictor is trained by maximizing the mutual information between the original predictions and new predictions. Note that even though the reparameterization trick is employed, the obtained masks are not strictly discrete but can largely alleviate the "introduced evidence" problem. In addition, since all edges in the dataset share the same predictor, the explanations can provide a global understanding of the trained GNNs.

- **GraphMask** [116] is a post-hoc method for explaining the edge importance in each GNN layer. Similar to the PGExplainer, it trains a classifier to predict whether an edge can be dropped without affecting the original predictions. However, GraphMask obtains an edge mask for each GNN layer while PGExplainer only focuses the input space. In addition, to avoiding changing graph structures, the dropped edges are replaced by learnable baseline connections, which are vectors with the same dimensions as node embeddings. Note that binary Concrete distribution [117] and reparameterization trick is employed to approximate discrete masks. In addition, the classifier is trained using the whole dataset by minimizing a divergence term, which measures the difference between network predictions. Similar to PGExplainer, it can largely alleviate the "introduced evidence" problem and provide a global understanding of the trained GNNs.

- **ZORRO** [118] employ discrete masks to identify important input nodes and node features. Given an input graph, a greedy algorithm is used to select nodes or node features step by step.

For each step, ZORRO selects one node or one node feature with the highest fidelity score. Note that the objective function, fidelity score, measure how the new predictions match the original predictions of the model by fixing the selected nodes/features and replacing the others with random noise values. Since there is no training procedure involved, the non-differentiable limitation of discrete masks is avoided. In addition, by using hard masks, ZORRO is not suffered from the "introduced evidence" problem. However, the greedy mask selection algorithm may lead to local optimal explanations. In addition, the explanations may lack a global understanding since masks are generated for each graph individually.

- **Causal Screening** [119] studies the causal attribution of different edges in the input graph. It identifies an edge mask for the explanatory subgraph. The key idea of causal attribution is to study the change of predictions when adding an edge into the current explanatory subgraph, known as the causal effect. For each step, it studies the causal effects of different edges and selects one edge to add to the subgraph. Specifically, it employs the individual causal effect (ICE) to select edges, which measures mutual information (between the predictions of original graphs and the explanatory subgraphs) difference after adding different edges to the subgraph. Similar to ZORRO, Causal Screening is a greedy algorithm generating discrete masks without any training procedure. Hence, it is not suffered from the "introduced evidence" problem but may lack a global understanding and stuck in local optimal explanations.

- **GraphLime** [120] extends the LIME algorithm [121] to deep graph models and studies the importance of different node features for node classification tasks. Given a target node in the input graph, GraphLime considers its N-hop neighboring nodes and their predictions as its local dataset where a reasonable choice of N is the number of layers in the trained GNNs. Then a nonlinear surrogate model, Hilbert-Schmidt Independence Criterion (HSIC) Lasso [122], is employed to fit the local dataset. Note that HSIC Lasso is a kernel-based feature selection algorithm. Finally, based on the weights of different features in HSIC Lasso, it can select important features to explain the HSIC Lasso predictions. Those selected fea-

tures are regarded as the explanations of the original GNN prediction. However, GraphLime can only provide explanations for node features but ignore graph structures, such as nodes and edges, which are more important for graph data. In addition, GraphLime is proposed to explain node classification predictions but cannot be directly applied to graph classification models.

- **RelEx** [123] also studies the explainability of node classification models by combining the ideas of surrogate methods and perturbation-based methods. Given a target node and its computational graph (N-hop neighbors), it first obtains a local dataset by randomly sampling connected subgraphs from the computational graph and feeding these subgraphs to the trained GNNs to obtain their predictions. Specifically, starting from the target node, it randomly selects neighboring nodes in a BFS manner. Next, it employs a GCN model as the surrogate model to fit the local datasets. Note that different from LIME and GraphLime, the surrogate model in RelEx is not interpretable. After training, it further applies the aforementioned perturbation-based methods, such as generating soft masks or Gumbel-Softmax masks, to explain the predictions. Compared with GraphLime, it can provide explanations regarding important nodes. However, it contains multiple steps of approximations, such as using the surrogate model to approximate local relationships and using masks to approximate the edge importance, thus making the explanations less convincing and trustable. Furthermore, as perturbation-based methods can be directly employed to explain original deep graph models, then it is not necessary to build another non-interpretable deep model as the surrogate model to explain. It is also unknown how it can be applied for graph classification tasks.

- **PGM-Explainer** [112] builds a probabilistic graphical model to provide instance-level explanations for GNNs. The local dataset is obtained by random node feature perturbation. Specifically, given an input graph, each time PGM-Explainer randomly perturbs the node features of several random nodes within the computational graph. Then for any node in the

computational graph, PGM-Explainer records a random variable indicating whether its features are perturbed and its influence on the GNN predictions. By repeating such procedures multiple times, a local dataset is obtained. Note that the local dataset of PGM-Explainer contains node variables instead of different neighboring graph samples. Then it selects top dependent variables to reduce the size of the local dataset via the Grow-Shrink (GS) algorithm [124]. Finally, an interpretable Bayesian network is employed to fit the local dataset and to explain the predictions of the original GNN model. PGM-Explainer can provide explanations regarding graph nodes but ignore graph edges, which contain important graph topology information. In addition, different from GraphLime and RelEx, the PGM-Explainer can be used to explain both node classification and graph classification tasks.

- **LRP** [125, 89] extends the original LRP algorithm [66] to deep graph models. It decomposes the output prediction score to different node importance scores. The score decomposition rule is developed based on the hidden features and weights. For a target neuron, its score is represented as a linear approximation of neuron scores from the previous layer. Intuitively, the neuron with a higher contribution of the target neuron activation receives a larger fraction of the target neuron score. Specifically, the study [89] applies the ϵ-stabilized rule from the original LRP, while existing work [125] employs the z^+-rule to perform decomposition. To satisfy the conservative property, the adjacency matrix is treated as a part of GNN model in the post-hoc explanation phase so that it can be ignored during score distribution; otherwise, the adjacent matrix will also receive decomposed scores, thus making the conservative property invalid. Since LRP is directly developed based on the model parameters, its explanation results are more trustable. However, it can only study the importance of different nodes and cannot be applied to graph structures, such as subgraphs and graph walks, which is more important for understanding GNNs. In addition, such a algorithm requires a comprehensive understanding of the model structures, which limits its applications for non-expert users, such as interdisciplinary researchers.

- **Excitation BP** [115] shares a similar idea as the LRP algorithm but is developed based on the law of total probability. It defines that the probability of a neuron in the current layer is equal to the total probabilities it outputs to all connected neurons in the next layer. Then the score decomposition rule can be regarded as decomposing the target probability into several conditional probability terms. Note that the computation of Excitation BP is highly similar to the z^+-rule in LRP. Hence, it shares the same advantages and limitations as the LRP algorithm.

- **GNN-LRP** [126] studies the importance of different graph walks. It is more coherent to the deep graph neural networks since graph walks correspond to message flows when performing neighborhood information aggregation. The score decomposition rule is the high-order Taylor decomposition of model predictions. It is shown that the Taylor decomposition (at root zero) only contains T-order terms where T is the number of layers in the trained GNNs. Then each term corresponds to a T-step graph walk and can be regarded as its importance score. Since it is not possible to directly compute the high-order derivatives given by Taylor expansion, GNN-LRP also follows a back propagation procedure to approximate the T-order terms. Note that the back propagation computation in GNN-LRP is similar to the LRP algorithm. However, instead of distributing scores to nodes or edges, GNN-LRP distributes scores to different graph walks. It records the paths of the distribution processes from layer to layer. Those paths are considered as different walks and the scores are obtained from their corresponding nodes. While GNN-LRP has a solid theoretical background, the approximations in its computations may not be accurate. In addition, the computational complexity is high since each walk is considered separately. Furthermore, it is also challenging for non-experts to use, especially for interdisciplinary domains.

4.3 The Proposed SubgraphX

While most current methods for GNN explanations are invariably based on identifying important nodes or edges, we argue that identifying important subgraphs is more natural and may lead to

better explainability. In this work, we propose a novel approach, known as SubgraphX, to explain GNNs by exploring and identifying important subgraphs.

4.3.1 From Node and Edge to Subgraph Explanations

Unlike images and texts, graph data contain important structural information, which is highly related to the properties of graphs. For example, network motifs, which can be considered as graph substructures, are simple building blocks of complex networks and may determine the functionalities of graphs in many domains, such as biochemistry, ecology, neurobiology, and engineering [93, 91, 94, 127]. Hence, investigating graph substructures is a crucial step towards the reverse engineering and understanding of the underlying mechanisms of GNNs. In addition, subgraphs are more intuitive and human-intelligible [128].

While different methods are proposed to explain GNNs, none of them can directly provide subgraph-level explanations for individual input examples. The XGNN can obtain graph patterns to explain GNNs but its explanations are not input-dependent and less precise. The other methods, such as GNNExplainer and PGExplainer, may obtain subgraph-level explanations by combining nodes or edges to form subgraphs in a post-processing manner. However, the important nodes or edges in their explanations are not guaranteed to be connected. Meanwhile, since GNNs are very complex, node/edge importance cannot be directly converted to subgraph importance. Furthermore, these methods ignore the interactions among different nodes and edges, which may contain important information. Hence, in this work, we propose a novel method, known as SubgraphX, to directly study the subgraphs to provide explanations. The explanations of our SubgraphX are connected subgraphs, which are more human-intelligible. In addition, by incorporating Shapley values, our method can capture the interactions among different graph structures when providing explanations.

4.3.2 Explaining GNNs with Subgraphs

We first present a formal problem formulation. Let $f(\cdot)$ denote the trained GNNs to be explained. Without loss of generality, we introduce our proposed SubgraphX by considering $f(\cdot)$ as

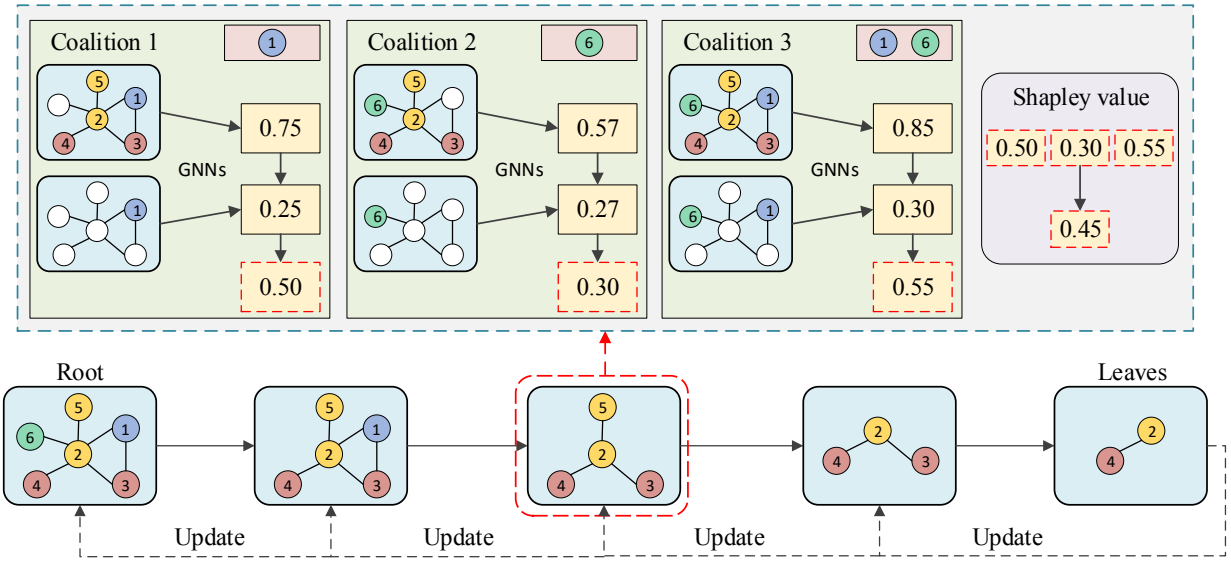

Figure 4.1: An illustration of our proposed SubgraphX. The bottom shows one selected path from the root to leaves in the search tree, which corresponds to one iteration of MCTS. For each node, its subgraph is evaluated by computing the Shapley value via Monte-Carlo sampling. In this example, we show the computation of Shapley value for the middle node (shown in red dashed box) where three coalitions are sampled to compute the marginal contributions. Note that nodes that are not selected are ignored for simplicity.

a graph classification model. Given an input graph $\mathcal{G}$, its predicted class is represented as y. The goal of our explanation task is to find the most important subgraph for the prediction y. Since disconnected nodes are hard to understand, we only consider connected subgraphs to enable the explanations to be more human-intelligible. Then the set of connected subgraphs of $\mathcal{G}$ is denoted as $\{\mathcal{G}_1, \cdots, \mathcal{G}_i, \cdots, \mathcal{G}_n\}$ where n is the number of different connected subgraphs in $\mathcal{G}$. The explanation of prediction y for input graph $\mathcal{G}$ can then be defined as

$$\mathcal{G}^* = \underset{|\mathcal{G}_i| \leq N_{\min}}{\operatorname{argmax}} \operatorname{Score}(f(\cdot), \mathcal{G}, \mathcal{G}_i), \tag{4.1}$$

where $\operatorname{Score}(\cdot, \cdot, \cdot)$ is a scoring function for evaluating the importance of a subgraph given the trained GNNs and the input graph. We use $N_{\min}$ as an upper bound on the size of subgraphs so that the obtained explanations are succinct enough. A straightforward way to obtain $\mathcal{G}^*$ is to enumerate all possible $\mathcal{G}_i$ and select the most important one as the explanation. However, such

a brute-force method is intractable when the graph is complex and large-scale. Hence, in this work, we propose to incorporate search algorithms to explore subgraphs efficiently. Specifically, we propose to employ Monte Carlo Tree Search (MCTS) [113, 129] as the search algorithm. In addition, since the information aggregation procedures in GNNs can be understood as interactions between different graph structures, we propose to employ the Shapley value [114] as the scoring function to measure the importance of different subgraphs by considering such interactions. We illustrate our proposed SubgraphX in Figure 4.1. After searching, the subgraph with the highest score is considered as the explanation of the prediction y for input graph $\mathcal{G}$. Note that our proposed SubgraphX can be easily extended to use other search algorithms and scoring functions.

4.3.3 Subgraph Exploration via MCTS

In our proposed SubgraphX, we employ the MCTS as the search algorithm to guide our subgraph explorations. We build a search tree in which the root is associated with the input graph and each of other nodes corresponds to a connected subgraph. Each edge in our search tree denotes that the graph associated with a child node can be obtained by performing node-pruning from the graph associated with its parent node. Formally, we define a node in this search tree as $\mathcal{N}_i$, and $\mathcal{N}_0$ denotes the root node. The edges in the search tree represent the pruning actions a. Note that each node may have many pruning actions, and these actions can be defined based on the dataset at hand or domain knowledge. Then the MCTS algorithm records the statistics of visiting counts and rewards to guide the exploration and reduce the search space. Specifically, for the node and pruning action pair $(\mathcal{N}_i, a_j)$, we assume that the subgraph $\mathcal{G}_j$ is obtained by action a_j from $\mathcal{G}_i$. Then the MCTS algorithm records four variables for $(\mathcal{N}_i, a_j)$, which are defined as:

- $C(\mathcal{N}_i, a_j)$ denotes the number of counts for selecting action a_j for node $\mathcal{N}_i$.
- $W(\mathcal{N}_i, a_j)$ is the total reward for all $(\mathcal{N}_i, a_j)$ visits.
- $Q(\mathcal{N}_i, a_j) = W(\mathcal{N}_i, a_j)/C(\mathcal{N}_i, a_j)$ and denotes the averaged reward for multiple visits.
- $R(\mathcal{N}_i, a_j)$ is the immediate reward for selecting a_j on $\mathcal{N}_i$, which is used to measure the importance of subgraph $\mathcal{G}_j$. We propose to use $R(\mathcal{N}_i, a_j) = \text{Score}(f(\cdot), \mathcal{G}, \mathcal{G}_j)$.

In each iteration, the MCTS selects a path starting from the root $\mathcal{N}_0$ to a leaf node $\mathcal{N}_\ell$. Note that the leaf nodes can be defined based on the numbers of nodes in subgraphs such that $|\mathcal{N}_\ell| \leq N_{\min}$. Formally, the action selection criteria of node $\mathcal{N}_i$ are defined as

$$a^* = \underset{a_j}{\operatorname{argmax}}\, Q(\mathcal{N}_i, a_j) + U(\mathcal{N}_i, a_j), \tag{4.2}$$

$$U(\mathcal{N}_i, a_j) = \lambda R(\mathcal{N}_i, a_j) \frac{\sqrt{\sum_k C(\mathcal{N}_i, a_k)}}{1 + C(\mathcal{N}_i, a_j)}, \tag{4.3}$$

where λ is a hyperparameter to control the trade-off between exploration and exploitation. In addition, $\sum_k C(\mathcal{N}_i, a_k)$ denotes the total visiting counts for all possible actions of node $\mathcal{N}_i$. Then the subgraph in the leaf node $\mathcal{N}_\ell$ is evaluated and the importance score is denoted as $\text{Score}(f(\cdot), \mathcal{G}, \mathcal{G}_\ell)$. Finally, all node and action pairs selected in this path are updated as

$$C(\mathcal{N}_i, a_j) = C(\mathcal{N}_i, a_j) + 1, \tag{4.4}$$

$$W(\mathcal{N}_i, a_j) = W(\mathcal{N}_i, a_j) + \text{Score}(f(\cdot), \mathcal{G}, \mathcal{G}_\ell). \tag{4.5}$$

After searching for several iterations, we select the subgraph with the highest score from the leaves as the explanation. Note that in early iterations, the MCTS tends to select child nodes with low visit counts in order to explore different possible pruning actions. In later iterations, the MCTS tends to select child nodes that yield higher rewards, *i.e.*, more important subgraphs.

4.3.4 A Game-Theoretical Scoring Function

In our proposed SubgraphX, both the MCTS rewards and the explanation selection are highly depending on the scoring function $\text{Score}(\cdot, \cdot, \cdot)$. It is crucial to properly measure the importance of different subgraphs. One possible solution is to directly feed the subgraphs to the trained GNNs $f(\cdot)$ and use the predicted scores as the importance scores. However, it cannot capture the interactions between different graph structures, thus affecting the explanation results. Hence, in this work, we propose to adopt the Shapley values [114, 30, 130] as the scoring function. The Shapley value is a solution concept from the cooperative game theory for fairly assigning a total game gain

to different game players. To apply it to graph model explanation tasks, we use the GNN prediction as the game gain and different graph structures as players.

Formally, given the input graph $\mathcal{G}$ with m nodes and the trained GNN $f(\cdot)$, we study the Shapley value for a target subgraph $\mathcal{G}_i$ with k nodes. Let $V = \{v_1, \cdots, v_i, \cdots, v_m\}$ denote all nodes in $\mathcal{G}$ and we assume that the nodes in $\mathcal{G}_i$ are $\{v_1, \cdots, v_k\}$ while the other nodes $\{v_{k+1}, \cdots, v_m\}$ belong to $\mathcal{G} \setminus \mathcal{G}_i$. Then the set of players is defined as $P = \{\mathcal{G}_i, v_{k+1}, \cdots, v_m\}$, where we consider the whole subgraph $\mathcal{G}_i$ as one player. Finally, the Shapley value of the player $\mathcal{G}_i$ can be computed as

$$\phi(\mathcal{G}_i) = \sum_{S \subseteq P \setminus \{\mathcal{G}_i\}} \frac{|S|!\,(|P| - |S| - 1)!}{|P|!} m(S, G_i), \tag{4.6}$$

$$m(S, \mathcal{G}_i) = f\left(S \cup \{\mathcal{G}_i\}\right) - f(S), \tag{4.7}$$

where S is the possible coalition set of players. Note that $m(S, \mathcal{G}_i)$ represents the marginalized contribution of player $\mathcal{G}_i$ given the coalition set S. It can be computed by the difference of predictions between incorporating $\mathcal{G}_i$ with and without the coalition set S. The obtained Shapley value $\phi(\mathcal{G}_i)$ considers all different coalitions to capture the interactions among different players. It is the only solution that satisfies four desirable axioms, including efficiency, symmetry, linearity, and dummy axiom [30], which can guarantee the correctness and fairness of the explanations. However, computing Shapley values using Eqs. (4.6) and (4.7) is time-consuming as it enumerates all possible coalitions, especially for large-scale and complex graphs. Hence, in this work, we propose to incorporate the GNN architecture information $f(\cdot)$ to efficiently approximate Shapley values.

4.3.5 Graph Inspired Efficient Computations

In graph neural networks, the new features of a target node are obtained by aggregating information from a limited neighboring region. Assuming there are L layers of GNN in the graph model $f(\cdot)$, then only the neighboring nodes within L-hops are used for information aggregation. Note that the information aggregation schema can be considered as interactions between different graph

structures. Hence, the subgraph $\mathcal{G}_i$ mostly interacts with the neighbors within L-hops. Based on such observations, we propose to compute the Shapley value of $\mathcal{G}_i$ by only considering its L-hop neighboring nodes. Specifically, assuming there are r ($r \leq m-k$) nodes within L-hop neighboring of subgraph $\mathcal{G}_i$, we denote these nodes as $\{v_{k+1}, \cdots, v_r\}$. Then the new set of players we need to consider is represented as $P' = \{\mathcal{G}_i, v_{k+1}, \cdots, v_r\}$. By incorporating P', the Shapley value of $\mathcal{G}_i$ can be defined as

$$\phi(\mathcal{G}_i) = \sum_{S \subseteq P' \setminus \{\mathcal{G}_i\}} \frac{|S|!\,(|P'| - |S| - 1)!}{|P'|!} m(S, G_i). \tag{4.8}$$

However, since graph data are complex that different nodes have variable numbers of neighbors, then P' may still contain a large number of players, thus affecting the efficiency of computation. Hence, in our SubgraphX, we further incorporate the Monte-Carlo sampling [131] to compute $\phi(\mathcal{G}_i)$. Specifically, for sampling step i, we sample a coalition set S_i from the player set $P' \setminus \{\mathcal{G}_i\}$ and compute its marginalized contribution $m(S_i, \mathcal{G}_i)$. Then the averaged contribution score for multiple sampling steps is regarded as the approximation of $\phi(\mathcal{G}_i)$. Formally, it can be mathematically written as

$$\phi(\mathcal{G}_i) = \frac{1}{T} \sum_{t=1}^{T} (f(S_i \cup \{\mathcal{G}_i\}) - f(S_i)), \tag{4.9}$$

where T is the total sampling steps. In addition, to compute the marginalized contribution, we follow a zero-padding strategy. Specifically, to compute $f(S_i \cup \{\mathcal{G}_i\})$, we consider the nodes $V \setminus (S_i \cup \{\mathcal{G}_i\})$ which are not belonging to the coalition or the subgraph and set their node features to all zeros. Then we feed the new graph to the GNNs $f(\cdot)$ and use the predicted probability as $f(S_i \cup \{\mathcal{G}_i\})$. Similarly, we can compute $f(S_i)$ by setting nodes $V \setminus S_i$ with zero features and feeding to the GNNs. It is noteworthy that we only perturb the node features instead of removing the nodes from the input graph because graphs are very sensitive to structural changes [132]. Finally, we conclude the computation steps of our proposed SubgraphX in Algorithm 1 and 2.

4.3.6 SubgraphX for Generic Graph Tasks

We have described our proposed SubgraphX using graph classification models as an example. It is noteworthy that our SubgraphX can be easily generalized to explain graph models on other

Algorithm 1 The algorithm of our proposed SubgraphX.

Input: GNN model $f(\cdot)$, input graph $\mathcal{G}$, MCTS iteration number M, the leaf threshold node number $N_{\min}$, $h(\mathcal{N}_i)$ denotes the associated subgraph of tree node $\mathcal{N}_i$.
Initialization: for each $(\mathcal{N}_i, a_j)$ pair , initialize its C, W, Q, and R variables as 0. The root of search tree is $\mathcal{N}_0$ associated with graph $\mathcal{G}$. The leaf set is set to $S_\ell = \{\}$.
for $i = 1$ **to** M **do**
  $curNode = \mathcal{N}_0$, $curPath = [\mathcal{N}_0]$
  while $h(curNode)$ has more node than $N_{\min}$ **do**
    for all possible pruning actions of $h(curNode)$ **do**
      Obtain child node $\mathcal{N}_j$ and its subgraph $\mathcal{G}_j$.
      Compute $R(curNode, a_j) = \text{Score}(f(\cdot), \mathcal{G}, \mathcal{G}_j))$ with Algorithm 2.
    end for
    Select the child $\mathcal{N}_{next}$ following Eq.(4.2, 4.3).
    $curNode = \mathcal{N}_{next}$, $curPath = curPath + \mathcal{N}_{next}$.
  end while
  $S_\ell = S_\ell \cup \{curNode\}$
  Update nodes in $curPath$ following Eq.(4.4, 4.5).
end for
Select subgraph with the highest score from S_ℓ.

tasks, such as node classification and link prediction. For node classification models, the explanation target is the prediction of a single node v_i given the input graph $\mathcal{G}$. Assuming there are L layers in the GNN models, the prediction of v_i only relies on its L-hop computation graph, denoted as $\mathcal{G}_c$. Then instead of searching from the input graph $\mathcal{G}$, our SubgraphX sets $\mathcal{G}_c$ as the corresponding graph of the search tree root $\mathcal{N}_0$. In addition, when computing the marginalized contributions, the zero-padding strategy should exclude the target node v_i. Meanwhile, for link prediction tasks, the explanation target is the prediction of a single link (v_i, v_j). Then the root of the search tree corresponds to the L-hop computation graph of node v_i and v_j. Similarly, the zero-padding strategy ignores the v_i and v_j when perturbing node features. Note that our SubgraphX treats the GNNs as black boxes during the explanation stage and only needs to access the inputs and outputs. Hence, our proposed SubgraphX can be applied to a general family of GNN models, including but not limited to GCNs [12], GATs [9], GINs [79], and Line-Graph NNs [133].

Algorithm 2 The algorithm of subgraph Shapley value.

Input: GNN model $f(\cdot)$ with L layers, input graph $\mathcal{G}$ with nodes $V = \{v_1, \ldots, v_m\}$, subgraph $\mathcal{G}_i$ with k nodes $\{v_1, \ldots, v_k\}$, Monte-Carlo sampling steps T.
Initialization: Obtain the L-hop neighboring nodes of $\mathcal{G}_i$, denoted as $\{v_{k+1}, \cdots, v_r\}$. Then the set of players is $P' = \{\mathcal{G}_i, v_{k+1}, \cdots, v_r\}$.
for $i = 1$ **to** T **do**
Sampling a coalition set S_i from $P' \setminus \{\mathcal{G}_i\}$.
Set nodes from $V \setminus (S_i \cup \{\mathcal{G}_i\})$ with zero features and feed to the GNNs $f(\cdot)$ to obtain $f(S_i \cup \{\mathcal{G}_i\})$.
Set nodes from $V \setminus S_i$ with zero features and feed to the GNNs $f(\cdot)$ to obtain $f(S_i)$.
Then $m(S_i, \mathcal{G}_i) = f(S_i \cup \{\mathcal{G}_i\}) - f(S_i)$.
end for
Return: Score$(f(\cdot), \mathcal{G}, \mathcal{G}_i) = \frac{1}{T}\sum_{t=1}^{T} m(S_i, \mathcal{G}_i)$.

4.4 Evaluating Explanation Techniques

Even though visualization results can provide an insightful understanding regarding whether the explanations are reasonable to humans, such evaluations are not fully trustable due to the lack of ground truths. In addition, to compare different explanation methods, humans need to study the results for each input example, which is time-consuming. Furthermore, human evaluations are highly dependent on their subjective understanding, which is not fair enough. Hence, evaluation metrics are crucial for studying explanation methods. Good metrics should evaluate the results from the model's perspective, such as whether the explanations are faithful to the model [134, 135]. In this section, we introduce several recently proposed evaluation metrics for explanation tasks.

4.4.1 Fidelity

First, the explanations should be faithful to the model. They should identify input features that are important for the model, not our humans. To evaluate this, the Fidelity+ [115] metric is recently proposed. Intuitively, if important input features (nodes/edges/node features) identified by explanation techniques are discriminative to the model, the predictions should change significantly when these features are removed. Hence, Fidelity+ is defined as the difference of accuracy (or predicted probability) between the original predictions and the new predictions after masking out

important input features [115, 136].

Formally, let $\mathcal{G}_i$ denote the i-th input graph and $f(\cdot)$ denote the GNN classifier to be explained. The prediction result of this graph is represented as $\hat{y}_i = \mathbf{argmax}\ f(\mathcal{G}_i)$. Then its explanations can be considered as a hard importance map m_i where each element is 0 or 1 to indicate if the corresponding feature is important. Note that for methods like ZORRO [118] and Causal Screening [119], the generated explanations are discrete masks, which can be directly used as the importance map. In addition, for methods like GNNExplainer [110] and GraphLime [120], the importance scores are continuous values, then the importance map m_i can be obtained by normalization and thresholding. Finally, the Fidelity+ score of prediction accuracy can be computed as

$$Fidelity{+}^{acc} = \frac{1}{N}\sum_{i=1}^{N}(\mathbb{1}(\hat{y}_i = y_i) - \mathbb{1}(\hat{y}_i^{1-m_i} = y_i)), \tag{4.10}$$

where y_i is the original prediction of graph i and N is the number of graphs. Here $1-m_i$ means the complementary mask that removes the important input features and $\hat{y}_i^{1-m_i}$ is the prediction when feeding the new graph into trained GNN $f(\cdot)$. The indicator function $\mathbb{1}(\hat{y}_i = y_i)$ returns 1 if $\hat{y}_i$ and y_i are equal and returns 0 otherwise. Note that the $Fidelity{+}^{acc}$ metric studies the change of prediction accuracy. By focusing on the predicted probability, the Fidelity+ of probability can be defined as

$$Fidelity{+}^{prob} = \frac{1}{N}\sum_{i=1}^{N}(f(\mathcal{G}_i)_{y_i} - f(\mathcal{G}_i^{1-m_i})_{y_i}), \tag{4.11}$$

where $\mathcal{G}_i^{1-m_i}$ represents the new graph obtained by keeping features of $\mathcal{G}_i$ based on the complementary mask $1-m_i$. Note that $Fidelity{+}^{prob}$ monitors the change of predicted probability, which is more sensitive than $Fidelity{+}^{acc}$. For both metrics, higher values indicate better explanations results and more discriminative features are identified.

The Fidelity+ metric studies the prediction change by removing important nodes/edges/node features. In contrast, the metric Fidelity- studies prediction change by keeping important input features and removing unimportant features. Intuitively, important features should contain discriminative information so that they should lead to similar predictions as the original predictions

even unimportant features are removed. Formally, the metric Fidelity- can be computed as

$$Fidelity{-}^{acc} = \frac{1}{N}\sum_{i=1}^{N}(\mathbb{1}(\hat{y}_i = y_i) - \mathbb{1}(\hat{y}_i^{m_i} = y_i)), \tag{4.12}$$

$$Fidelity{-}^{prob} = \frac{1}{N}\sum_{i=1}^{N}(f(\mathcal{G}_i)_{y_i} - f(\mathcal{G}_i^{m_i})_{y_i}), \tag{4.13}$$

where $\mathcal{G}_i^m$ is the new graph when keeping important features of $\mathcal{G}_i$ based on explanation m_i and $\hat{y}_i^{m_i}$ is the new prediction. Note that for both $Fidelity{-}^{acc}$ and $Fidelity{-}^{prob}$, lower values indicate less importance information are removed so that the explanations results are better.

4.4.2 Sparsity

Second, good explanations should be sparse, which means they should capture the most important input features and ignore the irrelevant ones. The metric Sparsity measures such a property. Specifically, it measures the fraction of features selected as important by explanation methods [115]. Formally, give the graph $\mathcal{G}_i$ and its hard importance map m_i, the Sparsity metric can be computed as

$$Sparsity = \frac{1}{N}\sum_{i=1}^{N}(1 - \frac{|m_i|}{|M_i|}), \tag{4.14}$$

where $|m_i|$ denotes the number of important input features (nodes/edges/node features) identified in m_i and $|M_i|$ means the total number of features in $\mathcal{G}_i$. Note that higher values indicate the explanations are more sparse and tend to only capture the most important input information.

4.4.3 Stability

In addition, good explanations should be stable. Intuitively, when small changes are applied to the input without affecting the predictions, the explanations should remain similar. The recent proposed Stability metric measures whether an explanation method is stable [137]. Given an input graph $\mathcal{G}_i$, its explanations m_i is regarded as the ground truth. Then the input graph $\mathcal{G}_i$ is perturbed by small changes, such as attaching new nodes/edges, to obtain a new graph $\hat{\mathcal{G}}_i$. Note that $\mathcal{G}_i$ and $\hat{\mathcal{G}}_i$ are required to have the same predictions. Then the explanations of $\hat{\mathcal{G}}_i$ is obtained, denoted

as $\hat{m}_i$. By comparing the difference between m_i and $\hat{m}_i$, we can compute the Stability score. Note that lower values indicate the explanation technique is more stable and more robust to noisy information. In addition, since graph representations are sensitive, selecting a proper amount of perturbations may be challenging.

4.4.4 Accuracy

Furthermore, the Accuracy metric is proposed for synthesis datasets [137, 110]. In synthesis datasets, even though it is unknown whether the GNNs make predictions in our expected way, the rules of building these datasets, such as graph motifs, can be used as reasonable approximations of the ground truths. Then for any input graph, we can compare its explanations with such ground truths. For example, when studying important edges, we can study the matching rate for important edges in explanations compared with those in the ground truths. The common metrics for such comparisons include general accuracy, F1 score, ROC-AUC score. Note that higher values indicate the explanations are closer to ground truths and can be considered as better results. In addition, the Accuracy metric cannot be applied to real-world datasets due to the lack of ground truths.

4.4.5 Discussions

It is noteworthy that different metrics should be combined to evaluate explanation results. For example, Sparsity and Fidelity+/Fidelity- are highly correlated. When the explanation results are soft values, the Sparsity is determined by a threshold value. Intuitively, larger threshold values tend to identify fewer features as important and hence increase the Sparsity score and decrease the Fidelity+ score. Hence, to fairly compare different explanation methods, we suggest comparing their Fidelity+ scores with the same level of Sparsity scores. One possible way is to select a fixed percentage of input features as important, and then compare their Fidelity+ or Fidelity- scores.

In addition, there are several other metrics to evaluate explanation results, such as Contrastivity [115] and Consistency [137]. The Contrastivity score is based on a strong assumption that the explanations for different classes should be significantly different, which ignores the common patterns among different classes. Meanwhile, the Consistency metric assumes that high-performing

model architectures should have consistent explanations. However, different models may capture different relationships even though they achieve competitive performance, especially for large-scale and complex datasets. Hence, we believe such metrics may not be fair enough to evaluate explanation results and they are not listed as recommended metrics.

4.5 Experimental Studies

4.5.1 Datasets and Experimental Settings

We conduct extensive experiments on different datasets and GNN models to demonstrate the effectiveness of our proposed method. We evaluate our SubgraphX with five datasets for both graph classification and node classification tasks, including synthetic data, biological data, and text data. We summarize these datasets as below:

- MUTAG [138] and BBBP [139] are molecular datasets for graph classification tasks. In these datasets, each graph represents a molecule while nodes are atoms and edges are bonds. The labels are determined by the chemical functionalities of molecules.

- Graph-SST2 [128] is sentiment graph dataset for graph classification. It converts text sentences to graphs with Biaffine parser [140] that nodes denote words and edges represent the relationships between words. Note that node embeddings are initialized as the pre-trained BERT word embeddings [141]. Each graph is labeled by its sentiment, which can be positive or negative.

- BA-2Motifs is a synthetic graph classification dataset. Each graph contains a based graph generated by *BarabÃąsi-Albert* (BA) model, which is connected with a house-like motif or a five-node cycle motif. The graphs are labeled based on the type of motifs. All node embeddings are initialized as vectors containing all 1s.

- BA-Shape is a synthetic node classification dataset. Each graph contains a base BA graph and house-like five-node motifs. The node labels are determined by the memberships and locations of nodes. All node embeddings are initialized as vectors containing all 1s.

Table 4.1: Statistics and properties of five datasets.

	Dataset				
	MUTAG	BBBP	GRAPH-SST2	BA-2MOTIFS	BA-SHAPE
# of Edges (avg)	19.79	25.95	9.20	25.48	2055
# of Nodes (avg)	17.93	24.06	10.19	25.0	700
# of Graphs	188	2039	70042	1000	1
# of Classes	2	2	2	2	4

We report the statistics and properties of the datasets in Table 4.1. We explore three variants of GNNs on these datasets, including GCNs, GATs, and GINs. All GNN models used in our experimental studies are trained to obtain reasonable performance. Then we compare our SubgraphX with several baselines, including MCTS_GNN, GNNExplainer [110], PGExplainer [111]. Here MCTS_GNN denotes the method using MCTS to explore subgraphs but directly employing the GNN predictions of these subgraphs as the scoring function. Specifically, we report the architectures and performance of these GNNs as below:

- **MUTAG (GCNs)**: This GNN model consists of 3 GCN layers. The input feature dimension is 7 and the output dimensions of different GCN layers are set to 128, 128, 128, respectively. We employ max-pooling as the readout function and ReLU as the activation function. The model is trained for 2000 epochs with a learning rate of 0.005 and the testing accuracy is 0.92. We study the explanations for the whole dataset.

- **MUTAG (GINs)**: This GNN model consists of 3 GIN layers. For each GIN layer, the MLP for feature transformations is a two-layer MLP. The input feature dimension is 7 and the output dimensions of different GIN layers are set to 128, 128, 128 respectively. We employ max-pooling as the readout function and ReLU as the activation function. The model is trained for 2000 epochs with a learning rate of 0.005 and the testing accuracy is 1.00. We study the explanations for the whole dataset.

- **BBBP (GCNs)**: This GNN model consists of 3 GCN layers. The input feature dimension is

9 and the output dimensions of different GCN layers are set to 128, 128, 128, respectively. We employ max-pooling as the readout function and ReLU as the activation function. The model is trained for 800 epochs with a learning rate of 0.005 and the testing accuracy is 0.863. We randomly split this dataset into the training set (80%), validation set (10%), and testing set (10%). We study the explanations for the testing set.

- **Graph-SST2 (GATs)**: This GNN model consists of 3 GAT layers. The input feature dimension is 768 and all GAT layers have 10 heads with 10-dimensional features. We employ max-pooling as the readout function and ReLU as the activation. In addition, we set the dropout rate to 0.6 to avoid overfitting. The model is trained for 800 epochs with a learning rate of 0.005 and the testing accuracy is 0.881. We follow the training, validation, and testing splitting of the original SST2 dataset. We study the explanations for the testing set.

- **BA-2Motifs (GCNs)**: This GNN model consists of 3 GCN layers. The input feature dimension is 10 and the output dimensions of different GCN layers are set to 20, 20, 20, respectively. For each GCN layer, we employ L2 normalization to normalize node features. We employ average pooling as the readout function and ReLU as the activation function. The model is trained for 800 epochs with a learning rate of 0.005 and the testing accuracy is 0.99. We randomly split this dataset into the training set (80%), validation set (10%), and testing set (10%). We study the explanations for the testing set.

- **BA-Shape (GCNs)**: This GNN model consists of 3 GCN layers. The input feature dimension is 10 and the output dimensions of different GCN layers are set to 20, 20, 20, respectively. For each GCN layer, we employ L2 normalization to normalize node features. In addition, we use ReLU as the activation function. The model is trained for 800 epochs with a learning rate of 0.005 and the testing accuracy is 0.957. We randomly split this dataset into the training set (80%), validation set (10%), and testing set (10%). We study the explanations for the testing set.

We conduct our experiments using one Nvidia V100 GPU on an Intel Xeon Gold 6248 CPU. Our

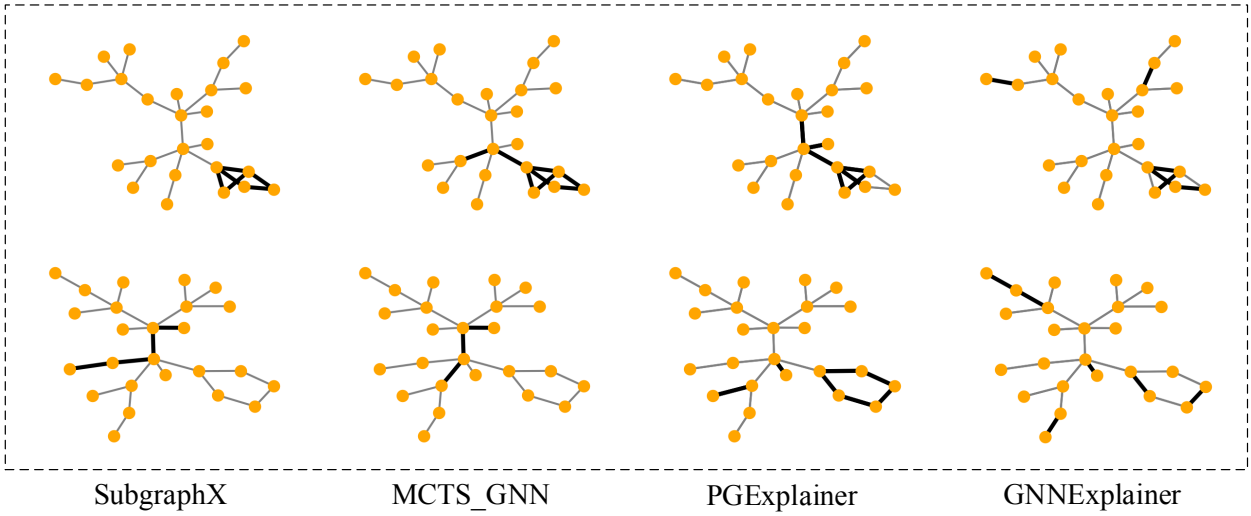

Figure 4.2: Explanation results on the BA-2Motifs dataset with a GCN graph classifier. The first row shows explanations for a correct prediction and the second row reports the results for an incorrect prediction.

implementations are based on Python 3.7.6, PyTorch 1.6.0, and Torch-geometric 1.6.3. For our proposed SubgraphX and other algorithms with MCTS, the MCTS iteration number M is set to 20. To explore a suitable trade-off between exploration and exploitation, we set the hyperparameter λ in Eq.(4.3) to 5 for Graph-SST2 (GATs) and BBBP (GCNs) models, and 10 for other models. Since all GNN models contain 3 network layers, we consider 3-hop computational graphs to compute Shapley values for our SubgraphX. For the Monte-Carlo sampling in our SubgraphX, we set the Monte-Carlo sampling steps T to 100 for all datasets. For MCTS†, we set Monte-Carlo sampling steps to 1000 to obtain good approximations since it samples from all nodes in a graph.

4.5.2 Explanations for Graph Classification Models

We first visually compare our SubgraphX with the other baselines using graph classification models. The results are reported in Figure 4.2, 4.3, and 4.4 where important substructures are shown in the bold.

The explanation results of the BA-2Motifs dataset are visualized in Figure 4.2. We use the GCNs as the graph classifier and report explanations for both correct and incorrect predictions. Since it is a synthetic dataset, we may consider the motifs as reasonable approximations of ex-

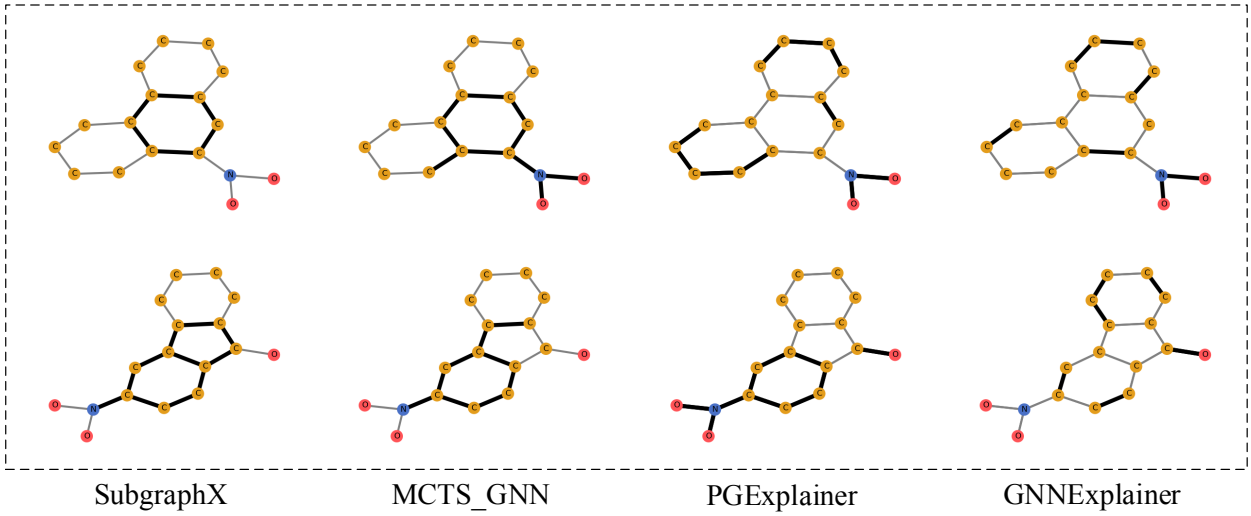

Figure 4.3: Explanation results on the MUTAG dataset with a GIN graph classifier. We show the explanations for two correct predictions. Here Carbon, Oxygen, and Nitrogen are shown in yellow, red, and blue, respectively.

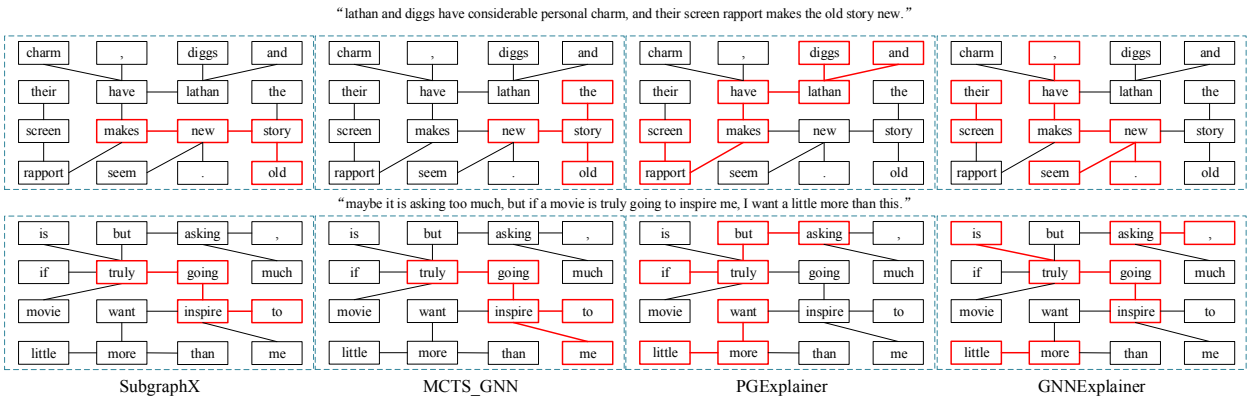

Figure 4.4: Explanation results on the Graph-SST2 dataset with a GAT graph classifier. The input sentences are shown on the top of explanations. Note that some "unimportant" words are ignored for simplicity. The first row shows explanations for a correct prediction and the second row reports the results for an incorrect prediction.

planation ground truth. In the first row, the model prediction is correct and our SubgraphX can precisely identify the house-like motif as the most important subgraph. In the second row, our SubgraphX explains the incorrect prediction that the GNN model cannot capture the five-node cycle motif as the important structure, and hence the prediction is wrong. For both cases, our SubgraphX can provide better visual explanations. In addition, our explanations are connected

subgraphs while PGExplainer and GNNExplainer identify discrete edges.

We also show the explanation results of the MUTAG dataset in Figure 4.3. Note that GINs are employed as the graph classification model to be explained. Since the MUTAG dataset is a real-world dataset and there is no ground truth for explanations, we evaluate the explanation results based on chemical domain knowledge. The graphs in MUTAG are labeled based on the mutagenic effects on a bacterium. It is known that carbon rings and NO_2 groups tend to be mutagenic [138]. In both examples, the predictions are "mutagenic" and our SubgraphX successfully and precisely identifies the carbon rings as important subgraphs. Meanwhile, the MCTS_GNN can capture the key subgraphs but include several additional edges. The results of the PGExplainer and GNNExplainer still contain several discrete edges.

For the dataset Graph-SST2, we employ GATs as the graph model and report the results in Figure 4.4. In the first row, the prediction is correct and the label is positive. Both our SubgraphX and the MCTS_GNN can find word phrases with positive semantic meaning, such as "makes old story new", which can reasonably explain the prediction. The explanations provided by PGExplainer and GNNExplainer are, however, less semantically related. In the second row, the input is negative but the prediction is positive. All methods except PGExplainer can explain the decision that the GNN model regards positive phrases "truly going to inspire" as important, thus yielding a positive but incorrect prediction. It is noteworthy that our method tends to include fewer neural words, such as "the", "me", and "screen", etc. Overall, our SubgraphX can explain both correct and incorrect predictions for different graph data and GNN models. Our explanations are more human-intelligible than comparing methods. More results for graph classification models are reported in Figure 4.5 and 4.6. In Figure 4.5, we show the explanations of real-world datasets BBBP and MUTAG. Obviously, our proposed method can provide more human-intelligible subgraphs as explanations while PGExplainer and GNNExplainer focus on discrete edges. In addition, we also report the results of sentiment dataset Graph-SST2 in Figure 4.6. The results show that our SubgraphX can provide reasonable explanations to explain the predictions. For example, in the second row, the input sentence is "none of this violates the letter of behan's book, but missing is its

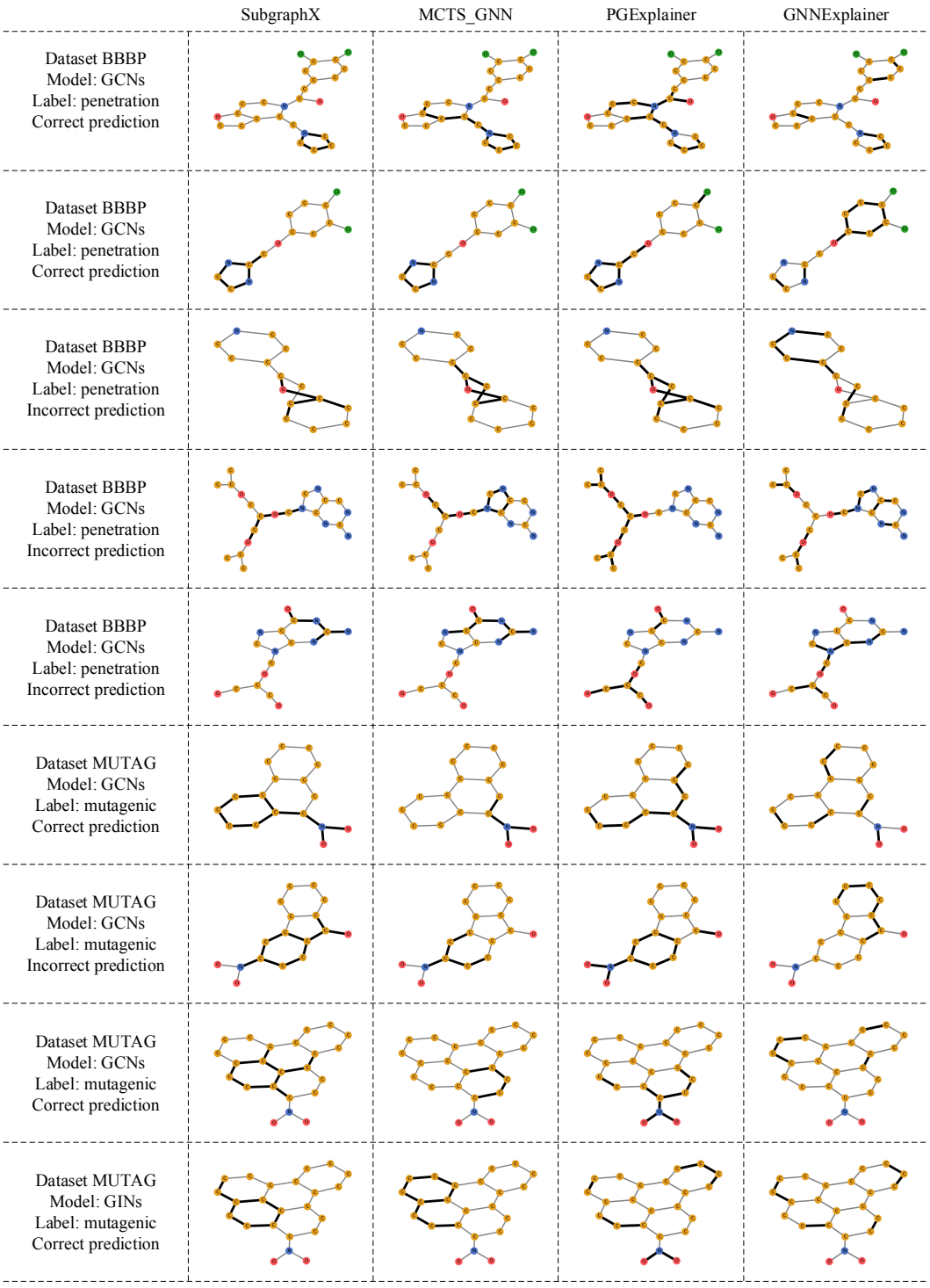

Figure 4.5: Explanation results of the BBBP and MUTAG datasets. Here Carbon, Oxygen, Nitrogen, and Chlorine are shown in yellow, red, and blue, green respectively.

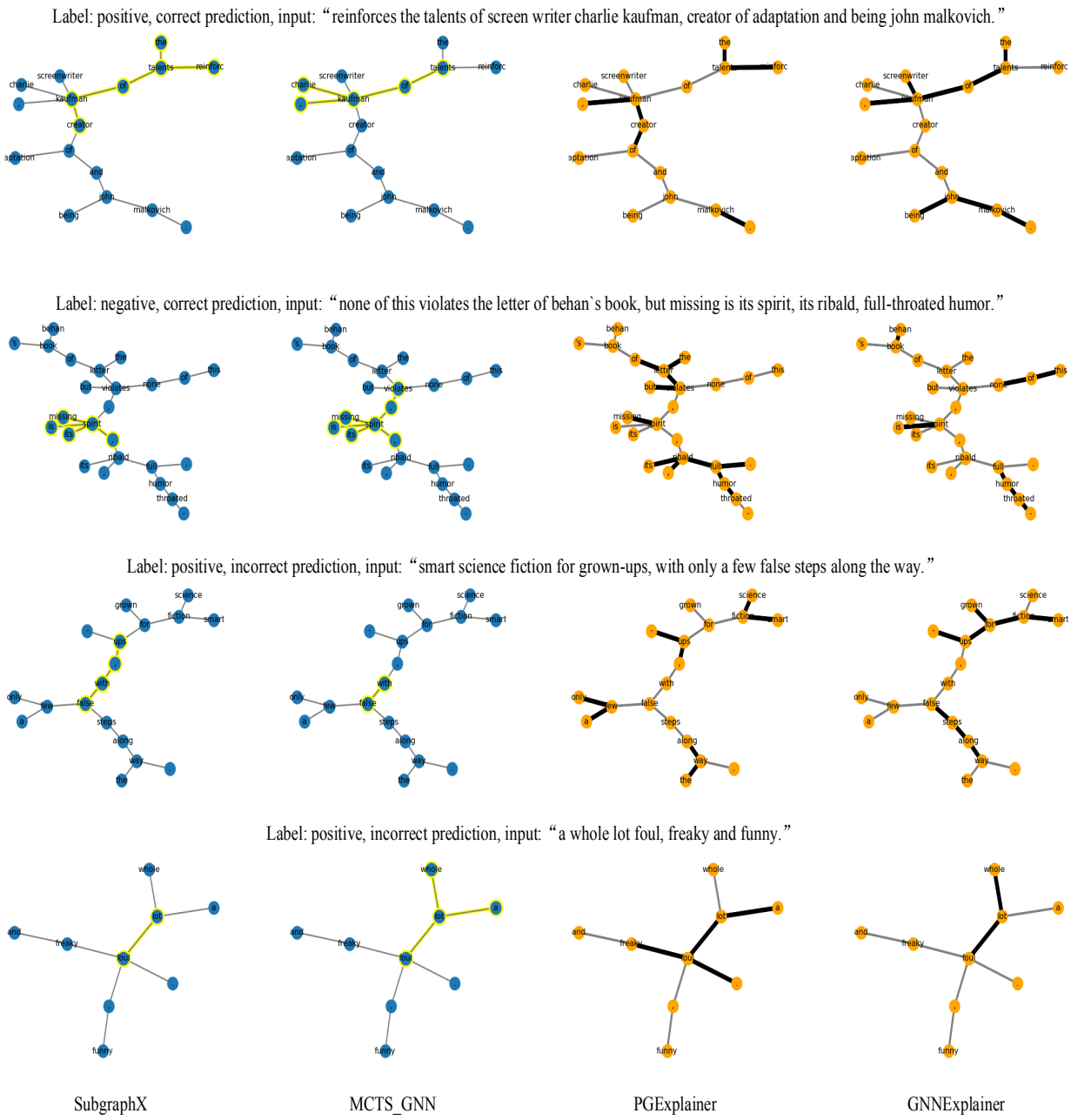

Figure 4.6: Explanation results of Grpah-SST2 dataset.

spirit, its ribald, full-throated humor”, whose label is negative and the prediction is correct. From the human’s view, “missing” should be the keyword for the semantic meaning. Our SubgraphX shows that the “missing is its spirit” phrase is important, which successfully captures the keyword. The other methods capture the words and phrases such as “violates”, “none of this”, which are less

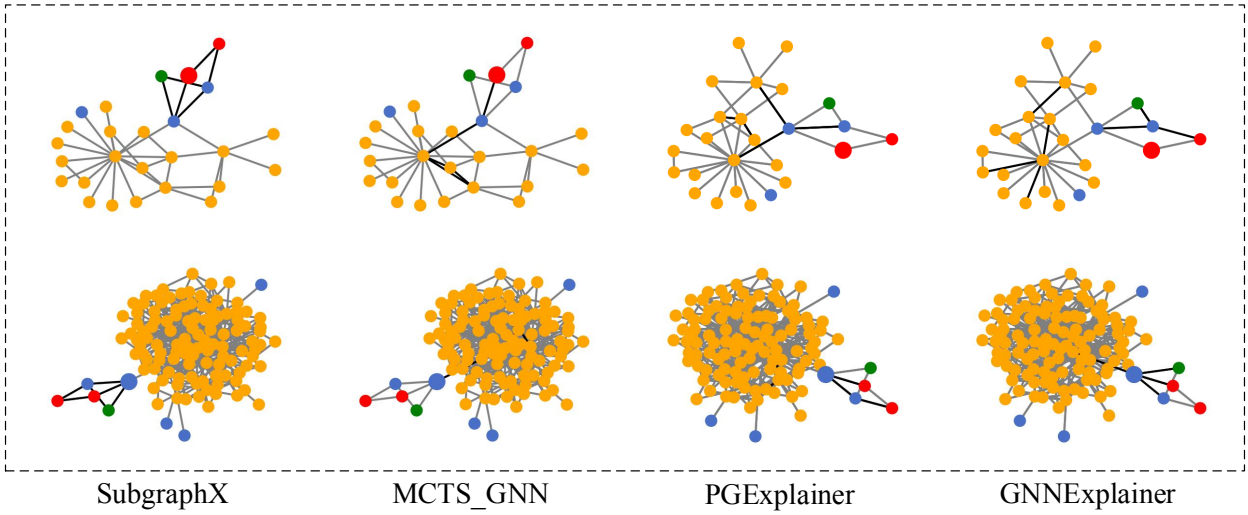

Figure 4.7: Explanation results on the BA-Shape dataset. The target node is shown in a larger size. Different colors denote node labels.

related to the negative meaning.

4.5.3 Explanations for Node Classification Models

We also compare different methods on the node classification tasks. We use the BA-Shape dataset and train a GCN model to perform node classification. The visualization results are reported in Figure 4.7 where the important substructures are shown in bold. We can verify if the explanations are consistent with the rules (the motifs) to label different nodes. For both examples, the target nodes are correctly classified. Obviously, our SubgraphX is precisely targeting the motifs as the explanations, which is reasonable and promising. For other methods, their explanations only cover partial motifs and include other structures. More visualization results of explanations for node classification models are reported in Figure 4.8 where we show the explanations of node classification dataset BA-Shape. Obviously, our SubgprahX focuses on the whole motifs for correct predictions and captures partial motifs for incorrect predictions. This is reasonable since if the model can capture the whole motif, then it is expected to correctly predict the target node; otherwise, the information of partial motifs is not enough to make correct predictions.

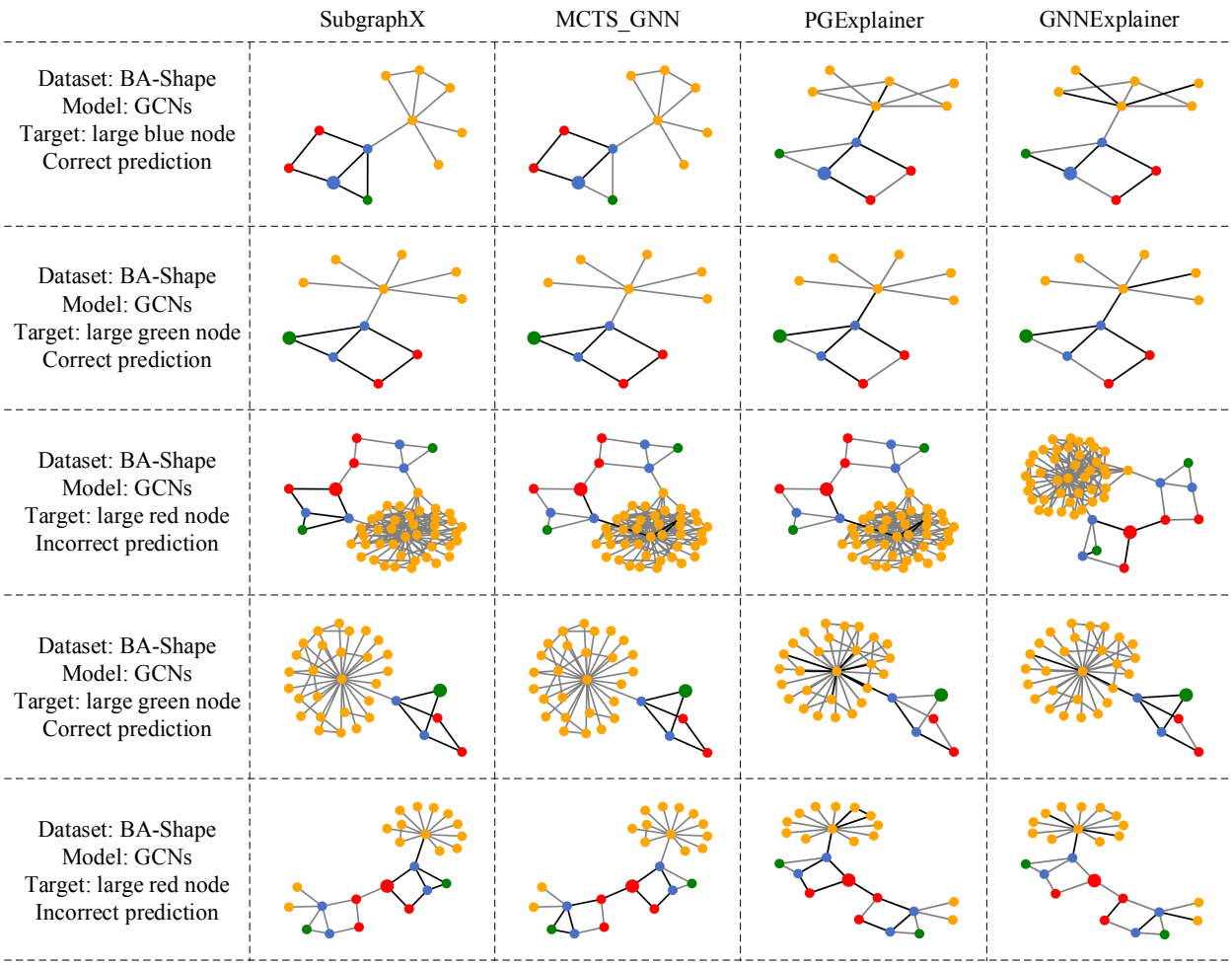

Figure 4.8: Explanation results of BA-Shape dataset. The target node is shown in a larger size.

4.5.4 Quantitative Studies

While visualizations are important to evaluate different explanation methods, human evaluations may not be accurate due to the lack of ground truths. Hence, we further conduct quantitative studies to compare these methods. Specifically, we employ the metrics $Fidelity{+}^{acc}$ and Sparsity discussed in Section 4.4 to evaluate explanation results. For simplicity, we use Fidelity to represent the $Fidelity{+}^{acc}$. The Fidelity metric measures whether the explanations are faithfully important to the model's predictions. It removes the important structures from the input graphs and computes the difference between predictions. In addition, the Sparsity metric measures the fraction of structures that are identified as important by explanation methods. Note that high Sparsity scores mean

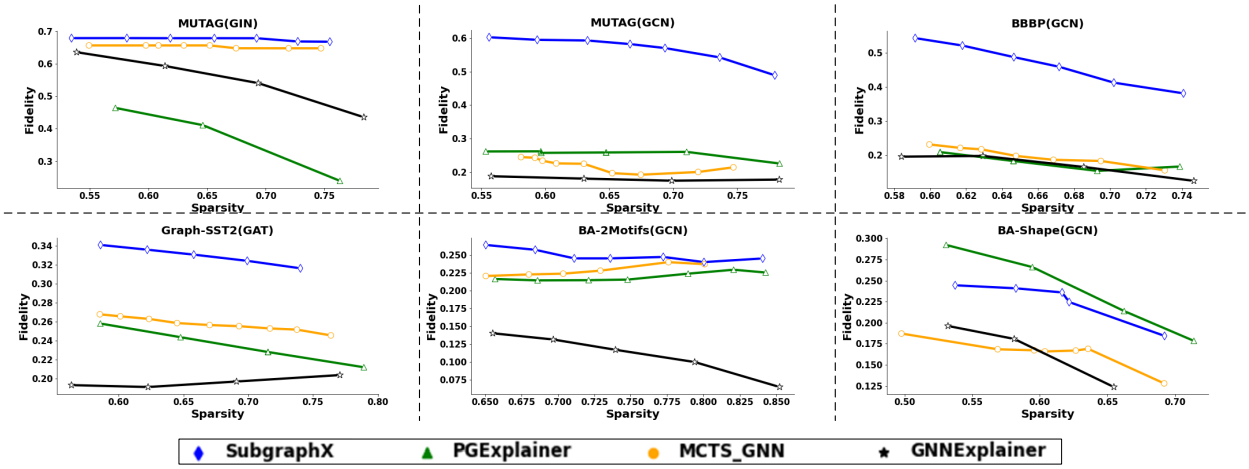

Figure 4.9: The quantitative studies for different explanation methods. Note that since the Sparsity scores cannot be fully controlled, we compare different methods with Fidelity scores under similar similar levels of Sparsity.

Table 4.2: Efficiency studies of different methods.

Method	**MCTS***	**MCTS**†	**SubgraphX**	**GNNExplainer**	**PGExplainer**
TIME	>10 hours	865.4 ± 1.6s	77.8 ± 3.8s	16.2 ± 0.2s	0.02s (Training 362s)
FIDELITY	N/A	0.53	0.55	0.19	0.18

smaller structures are identified as important, which can affect the Fidelity scores since smaller structures (high Sparsity) tend to be less important (low Fidelity). Hence, for fair comparisons, we compare different methods using Fidelity under similar levels of Sparsity. The results are reported in Figure 4.9 where we plot the curves of Fidelity scores with respect to the Sparsity scores. Obviously, for five out of six experiments, our proposed method outperforms the comparing methods significantly and consistently under different sparsity levels. For the BA-Shape (GCN) experiment, our SubgraphX obtains slightly lower but still competitive Fidelity scores compared with the PGExplainer. Overall, such results indicate that the explanations of our method are more faithful and important to the GNN models.

Table 4.3: The studies of different pruning strategies.

Method	Time	Fidelity
LOW2HIGH	107.24s	0.66149
HIGH2LOW	21.52s	0.61046

4.5.5 Efficiency Studies

Finally, we study the efficiency of our proposed method. For 50 graphs with an average of 24.96 nodes from the BBBP dataset, we show the averaging time cost to obtain explanations for each graph. We repeat the experiments 3 times and report the results Table 4.2. Here MCTS* denotes the baseline that follows Eq. (4.8) to compute Shapley values. Compared with our SubgraphX, the difference is the usage of Monte Carlo sampling. In addition, MCTS† indicates the baseline computing Shapley values with Monte Carlo sampling but without our proposed approximation schemes. Specifically, MCTS† samples coalition sets from the player set P instead of the reduced set P'. First, the time cost of MCTS* is extremely high since it needs to enumerate all possible coalition sets. Next, compared with MCTS†, our SubgraphX is 11 times faster while the obtained explanations have similar Fidelity scores. It demonstrates our approximation schemes are both effective and efficient. Even though our method is slower than GNNExplainer and PGExplainer, the Fidelity scores of our explanations are 300% higher than theirs. Furthermore, the PGExplainer requires to train its model using the whole dataset, which introduces the additional and significant time cost. Considering our explanations are with higher-quality and more human-intelligible, we believe such time complexity is reasonable and acceptable.

4.6 The Study of Pruning Actions

Finally, we discuss the pruning actions in our MCTS. For the graph associated with each non-leaf tree search node, we perform node pruning to obtain its children subgraphs. Specifically, when a node is removed, all edges connected with it are also removed. In addition, if multiple disconnected subgraphs are obtained after removing a node, only the largest subgraph is kept.

Instead of exploring all possible node pruning actions, we explore two strategies: Low2high and High2low. First, Low2high arranges the nodes based on their node degrees from low to high and only considers the pruning actions corresponding to the first k low degree nodes. Meanwhile, High2low arranges the nodes in order from high degree to low degree and only considers the first k high degree nodes for pruning. Intuitively, High2low is more efficient but may ignore the optimal solutions. In this work, we employ the High2low strategy for BA-Shape(GCNs), and Low2high strategy for other models, and set the k to 12 for all the datasets.

We conduct experiments to analyze these two pruning strategies for our SubgraphX algorithm and show the average time cost and Fidelity score in Table 4.3. Specifically, we randomly select 50 graphs from the BBBP datasets with an average node number of 24.96, which is the same in Section 4.5.5. In addition, we set Monte-Carlo sampling steps T to 100, and select the subgraphs with the highest Shapley values and contain less than 15 nodes to calculate the Fidelity. Obviously, High2low is 5 times faster than Low2high but the Fidelity scores of its explanations are inferior.

5. EXPLAINING DEEP GRAPH CLASSIFIER VIA GRAPH GENERATION*

Finally, we further explore the explainability of deep graph classifiers. Different from our proposed SubgraphX, in this chapter, we propose a novel method, known as XGNN, to provide model-level explanations for deep graph models.

5.1 Introduction

Graph Neural Networks (GNNs) have shown their effectiveness and obtained the state-of-the-art performance on different graph tasks, such as node classification [78, 9], graph classification [79, 80], and link prediction [81]. In addition, extensive efforts have been made towards different graph operations, such as graph convolution [12, 82, 83], graph pooling [8, 84], and graph attention [9, 85, 86]. Since graph data widely exist in different real-world applications, such as social networks, chemistry, and biology, GNNs are becoming increasingly important and useful. Despite their great performance, GNNs share the same drawback as other deep learning models; that is, they are usually treated as black-boxes and lack human-intelligible explanations. Without understanding and verifying the inner working mechanisms, GNNs cannot be fully trusted, which prevents their use in critical applications pertaining to fairness, privacy, and safety [87, 88]. For example, we can train a GNN model to predict the effects of drugs where we treat each drug as a molecular graph. Without exploring the working mechanisms, we do not know what chemical groups in a molecular graph lead to the predictions. Then we cannot verify whether the rules of the GNN model are consistent with real-world chemical rules, and hence we cannot fully trust the GNN model. This raises the need of developing explanation techniques for GNNs.

Recently, several explanation techniques have been proposed to explain deep learning models on image and text data. Depending on what kind of explanations are provided, existing techniques can be categorized into instance-level [18, 19, 20, 21, 22, 23, 24] or model-level [14, 15, 16] meth-

*Reprinted with permission from "XGNN: Towards Model-Level Explanations of Graph Neural Networks" by Hao Yuan, Jiliang Tang, Xia Hu, and Shuiwang Ji, *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, vol. 1, pp. 430-438, Copyright 2020 by ACM.

ods. instance-level explanations explain the prediction for a given input example, by determining important features in the input or the decision procedure for this input through the model. Common techniques in this category include gradient-based methods [18, 19], visualizations of intermediate feature maps [20, 21], and occlusion-based methods [22, 23, 24]. Instead of providing input-dependent explanations, model-level explanations aim to explain the general behavior of the model by investigating what input patterns can lead to a certain prediction, without respect to any specific input example. Input optimization [14, 15, 16, 25] is the most popular model-level explanation method. These two categories of explanation methods aim at explaining deep models in different views. Since the ultimate goal of explanations is to verify and understand deep models, we need to manually check the explanation results and conclude if the deep models work in our expected way. For instance-level methods, we may need to explore the explanations for a large number of examples before we can trust the models. However, it is time-consuming and requires extensive expert efforts. For model-level methods, the explanations are more general and high-level, and hence need less human supervision. However, the explanations of model-level methods are less precise compared with instance-level explanations. Overall, both model-level and instance-level methods are important for explaining and understanding deep models.

Explaining deep learning models on graph data become increasingly important but is still less explored. To the best of our knowledge, there is no existing study on explaining GNNs at the model-level. The existing study [88, 89] only provides instance-level explanations for graph models. As a radical departure from existing work, we propose a novel explanation technique, known as XGNN, for explaining deep graph models at the model-level. We propose to investigate what graph patterns can maximize a certain prediction. Specifically, we propose to train a graph generator such that the generated graph patterns can be used to explain deep graph models. We formulate it as a reinforcement learning problem that at each step, the graph generator predicts how to add an edge to a given graph and form a new graph. Then the generator is trained based on the feedback from the trained graph models using policy gradient [58]. We also incorporate several graph rules to encourage the generated graphs to be valid. Note that the graph generation part in our XGNN

framework can be generalized to any suitable graph generation method, determined by the dataset at hand and the GNNs to be explained. Finally, we trained GNN models on both real-world and synthetic datasets which can yield good performance. Then we employ our proposed XGNN to explain these trained models. Experimental results show that our proposed XGNN can find the desired graph patterns and explains these models. With our generated graph patterns, we can verify, understand, and even improve the trained GNN models.

5.2 Related Work

5.2.1 Graph Neural Networks

Graphs are wildly employed to represent data in different real-world domains and graph neural networks have shown promising performance on these data. Different from image and text data, a graph is represented by a feature matrix and an adjacency matrix. Formally, a graph G with n nodes is represented by its feature matrix $X \in \mathbb{R}^{n\times d}$ and its adjacency matrix $A \in \{0,1\}^{n\times n}$. Note that we assume each node has a d-dimension vector to represent its features. Graph neural networks learn node features based on these matrices. Even though there are several variants of GNNs, such as graph convolution networks (GCNs) [12], graph attention networks (GATs) [9], and graph isomorphism networks (GINs) [79], they share a similar feature learning strategy. For each node, GNNs update its node features by aggregating the features from its neighbors and combining them with its own features. We take GCNs as an example to illustrate the neighborhood information aggregation scheme. The operation of GCNs is defined as

$$X_{i+1} = f(D^{-\frac{1}{2}}\hat{A}D^{-\frac{1}{2}}X_iW_i), \tag{5.1}$$

where $X_i \in \mathbb{R}^{n\times d_i}$ and $X_{i+1} \in \mathbb{R}^{n\times d_{i+1}}$ are the input and output feature matrices of the i^{th} graph convolution layer. In addition, $\hat{A} = A + I$ is used to add self-loops to the adjacency matrix, D denotes the diagonal node degree matrix to normalize $\hat{A}$. The matrix $W_i \in \mathbb{R}^{d_i\times d_{i+1}}$ is a trainable matrix for layer i and is used to perform linear feature transformation and $f(\cdot)$ denotes a non-linear activation function. By stacking j graph convolution layers, the j-hop neighborhood information

can be aggregated. Due to its superior performance, we incorporate the graph convolution in Equation (5.1) as our graph neural network operator.

5.2.2 Model-level Explanations

Next, we briefly discuss popular model-level explanation techniques for deep learning models on image data, known as input optimization methods [14, 15, 16, 25]. These methods generally generate optimized input that can maximize a certain behavior of deep models. They randomly initialize the input and iteratively update the input towards an objective, such as maximizing a class score. Then such optimized input can be regarded as the explanations for the target behavior. Such a procedure is known as optimization and is similar to training deep neural networks. The main difference is that in such input optimization techniques, all network parameters are fixed while the input is treated as trainable variables. While such methods can provide meaningful model-level explanations for deep models on images, they cannot be directly applied to explain GNNs due to three challenges. First, the structural information of a graph is represented by a discrete adjacency matrix, which cannot be directly optimized via back-propagation. Second, for images, the optimized input is an abstract image and the visualization shows high-level patterns and meanings. In the case of graphs, the abstract graph is not meaningful and hard to visualize. Third, the obtained graphs may not be valid for chemical or biological rules since non-differentiable graph rules cannot be directly incorporated into optimization. For example, the node degree of an atom should not exceed its maximum chemical valency.

5.2.3 Graph Model Explanations

To the best of our knowledge, there are only a few existing studies focusing on the explainability of deep graph models [88, 89]. The recent GNN explanation tool GNN Explainer [88] proposes to explain deep graph models at the instance-level by learning soft masks. For a given example, it applies soft masks to graph edges and node features and updates the masks such that the prediction remains the same as the original one.

Then some graph edges and node features are selected by thresholding the masks, and they are

treated as important edges and features for making the prediction for the given example. The other work [89] also focuses on the instance-level explanations of deep graph models. It applies several well-known image explanation methods to graph models, such as sensitivity analysis (SA) [90], guided backpropagation (GBP) [43], and layer-wise relevance propagation (LRP) [66]. The SA and GBP methods are based on the gradients while the LRP method computes the saliency maps by decomposing the output prediction into a combination of its inputs. In addition, both of these studies generate input-dependent explanations for individual examples. To verify and understand a deep model, humans need to check explanations for all examples, which is time-consuming or even not feasible.

While input-dependent explanations are important for understanding deep models, model-level explanations should not be ignored. However, none of the existing work investigates the model-level explanations of deep graph models. In this work, we argue that model-level explanations can provide higher-level insights and a more general understanding in how a deep learning model works. Therefore, we aim at providing model-level explanations for GNNs. We propose a novel method, known as XGNN, to explain GNNs by graph generation such that the generated graphs can maximize a certain behavior.

5.3 XGNN: Explainable Graph Neural Networks

5.3.1 Model-Level GNN Explanations

Intuitively, given a trained GNN model, the model-level explanations for it should explain what graph patterns or sub-graph patterns lead to a certain prediction. For example, one possible type of patterns is known as network motifs that represent simple building blocks of complex networks (graphs), which widely exist in graphs from biochemistry, neurobiology, ecology, and engineering [91, 92, 93, 94]. Different motif sets can be found in graphs with different functions [91, 92], which means different motifs may directly relate to the functions of graphs. However, it is still unknown whether GNNs make predictions based on such motifs or other graph information. By identifying the relationships between graph patterns and the predictions of GNNs, we can better

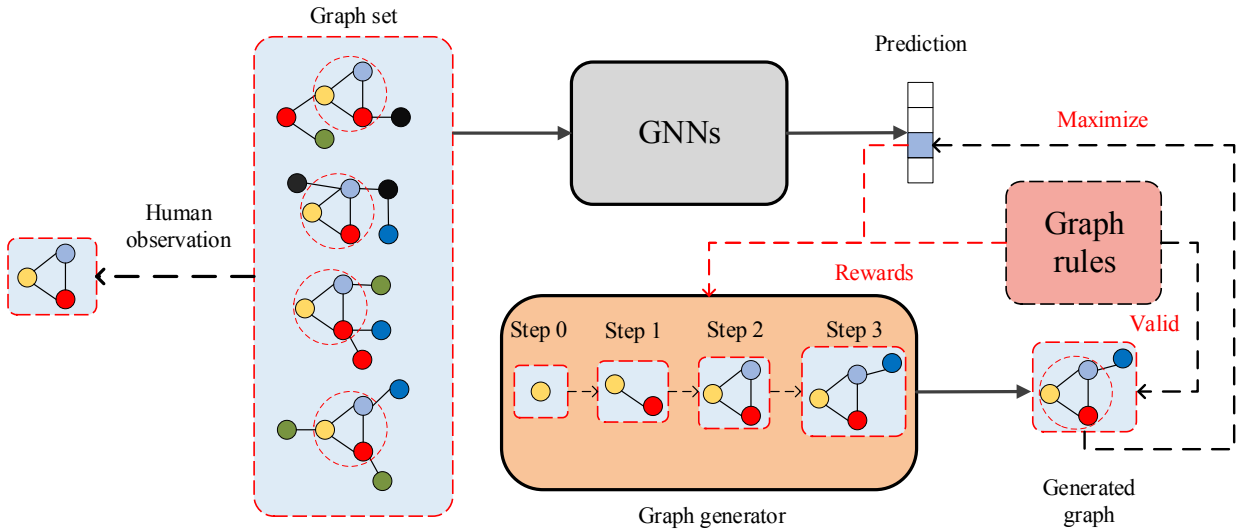

Figure 5.1: Illustrations of our proposed XGNN for graph explanations via graph generation. The GNNs represent a trained graph classification model that we try to explain. All graph examples in the graph set are classified to the third class. The left part shows that we can manually conclude the key graph patterns for the third class but it is challenging. The right part shows that we propose to train a graph generator to generate graphs that can maximize the class score and be valid according to graph rules.

understand the models and verify whether a model works as expected. Therefore, we propose our XGNN, which explains GNNs using such graph patterns. Specifically, in this chapter, we investigate the model-level explanations of GNNs for graph classification tasks and the graph patterns are obtained by graph generations.

Formally, let $f(\cdot)$ denote a trained GNN classification model, and $y \in \{c_1, \cdots, c_\ell\}$ denote the classification prediction. Given $f(\cdot)$ and a chosen class c_i, $i \in \{1, \cdots, \ell\}$, our goal is to investigate what input graph patterns maximize the predicted probability for this class. The obtained patterns can be treated as model-level explanations with respect to c_i. Formally, the task can be defined as

$$G^* = \underset{G}{\operatorname{argmax}}\, P(f(G) = c_i), \tag{5.2}$$

where G^* is the optimized input graph we need. A popular way to obtain such optimized input for explaining image and text models is known as input optimization [14, 15, 16, 25]. However, as

discussed in Section 5.2.2, such optimization method cannot be applied to explain graph models because of the special representations of graph data. Instead, we propose to obtain the optimized graph G^* via graph generation. The general illustration of our proposed method is shown in Figure 5.1. Given a pre-trained graph classification model, we explain it by providing explanations for its third class. We may manually conclude the graph patterns from the graph dataset. By evaluating all graph examples in the dataset, we can obtain the graphs that are predicted to be the third class. Then we can manually check what are the common graph patterns among these graphs. For example, the left part of Figure 5.1 shows that a set of four graphs are classified into the third class. Based on human observations, we know that the important graph pattern leading to the prediction is the triangle pattern consisting of a red node, a yellow node, and a blue node. However, such manual analysis is time-consuming and not applicable for large-scale and complex graph datasets. As shown in the right part, we propose to train a graph generator to generate graph patterns that can maximize the prediction score of the third class. In addition, we incorporate graph rules, such as the chemical valency check, to encourage valid and human-intelligible explanations. Finally, we can analyze the generated graphs to obtain model-level explanations for the third class. Compared with directly manual analysis on the original dataset, our proposed method generates small-scale and less complex graphs, which can significantly reduce the cost for further manual analysis.

5.3.2 Explaining GNNs via Graph Generation

Recent advances in graph generation lead to many successful graph generation models, such as GraphGAN [95], ORGAN [96], Junction Tree VAE [97], DGMG [98], and Graph Convolutional Policy Network (GCPN) [99]. Inspired by these methods, we propose to train a graph generator which generates G^* step by step. For each step, the graph generator generates a new graph based on the current graph. Formally, we define the partially generated graph at step t as G_t, which contains n_t nodes. It is represented as a feature matrix $X_t \in \mathbb{R}^{n_t \times d}$ and an adjacency matrix $A_t \in \{0,1\}^{n_t \times n_t}$, assuming each node has a d-dimensional feature vector. Then we define a θ-parameterized graph generator as $g_\theta(\cdot)$, which takes G_t as input, and outputs a new graph G_{t+1}

that

$$X_{t+1}, A_{t+1} = g_\theta(X_t, A_t). \tag{5.3}$$

Then the generator is trained with the guidance from the pre-trained GNNs $f(\cdot)$. Since generating the new graph G_{t+1} from G_t is non-differentiable, we formulate the generation procedure as a reinforcement learning problem. Specifically, assuming there are k types of nodes in the dataset, we define a candidate set $C = \{s_1, s_2, \cdots, s_k\}$ denoting these possible node types. For example, in a chemical molecular dataset, the candidate set may include Carbon, Nitrogen, Oxygen, Fluorine, etc. In a social network dataset where nodes are not labeled, the candidate set only contains a single node type. Then at each step t, based on the partially generated graph G_t, the generator $g(\cdot)$ generates G_{t+1} by predicting how to add an edge to the current graph G_t. Note that the generator may add an edge between two nodes in the current graph G_t or add a node from the candidate set C to the current graph G_t and connect it with an existing node in G_t. Formally, we formulate it as a reinforcement learning problem, which consists of four elements: state, action, policy, and reward.

State: The state of the reinforcement learning environment at step t is the partially generated graph G_t. The initial graph at the first step can be either a random node from the candidate set C or manually designed based on prior domain knowledge. For example, for the dataset describing organic molecules, we can set the initial graph as a single node labeled with carbon atom since any organic compound contains carbon generally [100].

Action: The action at step t, denoted as a_t, is to generate the new graph G_{t+1} based on the current graph G_t. Specifically, given the current state G_t, the action a_t is to add an edge to G_t by determining the starting node and the ending node of the edge. Note that the starting node $a_{t,start}$ can be any node from the current graph G_t while the ending node $a_{t,end}$ is selected from the union of the current graph G_t and the candidate set C excluding the selected starting node $a_{t,start}$, denoted as $(G_t \bigcup C) \setminus a_{t,start}$. Note that with the predefined maximum action step and maximum node number, we can control the termination of graph generation.

Policy: We employ graph neural networks to serve as the policy. The policy determines the action a_t based on the state G_t. Specifically, the policy is the graph generator $g_\theta(\cdot)$, which takes G_t

and C as the input and outputs the probabilities of possible actions. With the reward function, the generator $g_\theta(\cdot)$ can be trained via policy gradient [58].

Reward: The reward for step t, denoted as R_t, is employed to evaluate the action at step t, which consists of two parts. The first part is the guidance from the trained GNNs $f(\cdot)$, which encourages the generated graph to maximize the class score of class c_i. By feeding the generated graphs to $f(\cdot)$, we can obtain the predicted probabilities for class c_i and use them as the feedback to update $g_\theta(\cdot)$. The second part encourages the generated graphs to be valid in terms of certain graph rules. For example, for social network datasets, it is may not allowed to add multiple edges between two nodes. In addition, for chemical molecular datasets, the degree of an atom cannot exceed its chemical valency. Note that for each step, we include both intermediate rewards and overall rewards to evaluate the action.

While we formulate the graph generation as a reinforcement learning problem, it is noteworthy that our proposed XGNN is a novel and general framework for explaining GNNs at the model-level. The graph generation part in this framework can be generalized to any suitable graph generation method, determined by the dataset at hand and the GNNs to be explained.

5.3.3 Graph Generator

For step t, the graph generator $g_\theta(\cdot)$ incorporates the partially generated graph G_t and the candidate set C to predict the probabilities of different actions, denoted as $p_{t,start}$ and $p_{t,end}$. Assume there are n_t nodes in G_t and k nodes in C, then both $p_{t,start}$ and $p_{t,end}$ are with $n_t + k$ dimensionality. Then the action $a_t = (a_{t,start}, a_{t,end})$ is sampled from the probabilities $p_t = (p_{t,start}, p_{t,end})$. Next, we can obtain the new graph G_{t+1} based on the action a_t. Specifically, in our generator, we first employ several graph convolutional layers to aggregate neighborhood information and learn node features. Mathematically, it can be written as

$$\widehat{X} = \text{GCNs}(G_t, C), \tag{5.4}$$

where $\widehat{X}$ denotes the learnt node features. Note that the graph G_t and the candidate set C are combined as the input of GCNs. We merge all nodes in C to G_t without adding any edge and then obtain the new node feature matrix and adjacency matrix. Then Multilayer Perceptrons (MLPs) are used to predict the probabilities of the starting node, $p_{t,start}$ and the action $a_{t,start}$ is sampled from this probabilty distribution. Mathematically, it can be written as

$$p_{t,start} = \text{Softmax}(\text{MLPs}(\widehat{X})), \tag{5.5}$$

$$a_{t,start} \sim p_{t,start} \cdot m_{t,start}, \tag{5.6}$$

where $\cdot$ means element-wise product and $m_{t,start}$ is to mask out all candidate nodes since the starting node can be only selected from the current graph G_t. Let $\widehat{x}_{start}$ denote the features of the node selected by the start action $a_{t,start}$. Then conditioned on the selected node, we employ the second MLPs to compute the probability distribution of the ending node $p_{t,end}$ from which we sample the ending node action $a_{t,end}$. Note that since the starting node and the ending node cannot be the same, we apply a mask $m_{t,end}$ to mask out the node selected by $a_{t,start}$. Mathematically, it can be written as

$$p_{t,end} = \text{Softmax}(\text{MLPs}([\widehat{X}, \widehat{x}_{start}])), \tag{5.7}$$

$$a_{t,end} \sim p_{t,end} \cdot m_{t,end}, \tag{5.8}$$

where $[\cdot, \cdot]$ denotes broadcasting and concatenation. In addition, $m_{t,end}$ is the mask consisting of all 1s except the position indicating $a_{t,start}$. Note that the same graph generator $g_\theta(\cdot)$ is shared by different time steps, and our generator is capable to incorporate graphs with variable sizes.

We illustrate our graph generator in Figure 5.2 where we show the graph generation procedure for one step. The current graph G_t consists of 4 nodes and the candidate set has 3 available nodes. They are combined together to serve as the input of the graph generator. The embeddings of candidate nodes are concatenated to the feature matrix of G_t while the adjacency matrix of G_t is expanded accordingly. Then multiple graph convolutional layers are employed to learn features

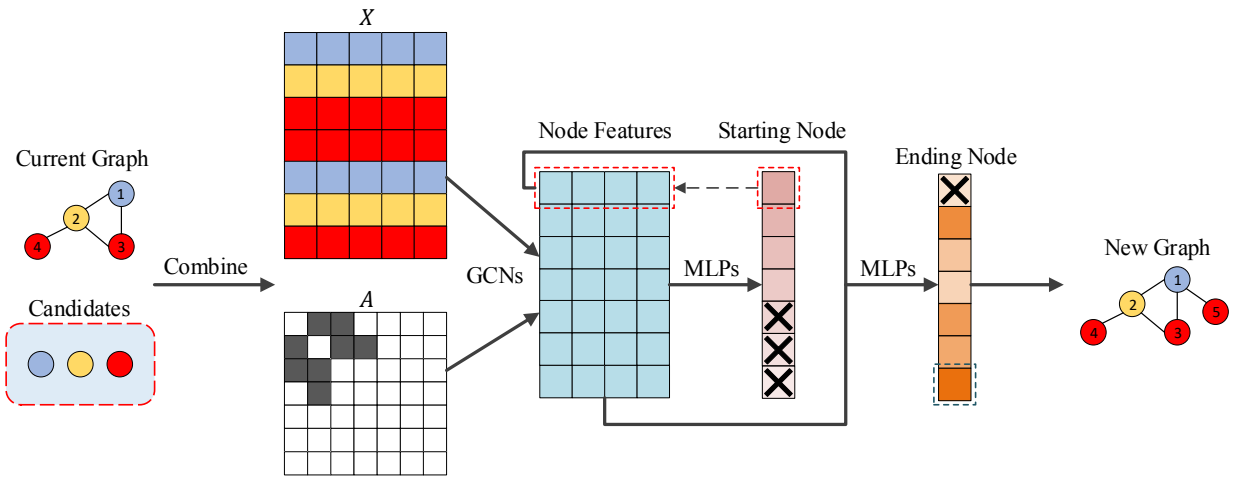

Figure 5.2: An Illustration of our graph generator for processing a single step. Different colors denote different types of node. Given a graph with 4 nodes and a candidate set with 3 nodes, we first combine them together to obtain the feature matrix and the adjacency matrix. Then we employ several GCN layers to aggregate and learn node features. Next, the first MLPs predict a probability distribution from which we sample the starting node. Finally, the second MLPs predict the ending node conditioned on the starting node. Note that the black crosses indicates masking out nodes.

for all nodes. With the first MLPs, we obtain the probabilities of selecting different nodes as the starting node, and from which we sample the node 1 as the starting node. Then based on the features of node 1 and all node features, the second MLPs predict the ending node. We sample from the probabilities and select the node 7 as the ending node, which corresponds to the red node in the candidate set. Finally, a new graph is obtained by including a red node and connecting it with node 1.

5.3.4 Training the Graph Generator

The graph generator is trained to generate specific graphs that can maximize the class score of class c_i and be valid to graph rules. Since such guidance is not differentiable, we employ policy gradient [58] to train the generator. According to [56, 57], the loss function for the action a_t at step t can be mathematically written as

$$\mathcal{L}_g = -R_t(\mathcal{L}_{CE}(p_{t,start}, a_{t,start}) + \mathcal{L}_{CE}(p_{t,end}, a_{t,end})), \tag{5.9}$$

where $\mathcal{L}_{CE}(\cdot,\cdot)$ denotes the cross entropy loss and R_t means the reward function for step t. Intuitively, the reward R_t indicates whether a_t has a large chance to generate graph with high class score of class c_i and being valid. Hence, the reward R_t consists of two parts. The first part $R_{t,f}$ is the feedback from the trained model $f(\cdot)$ and the second part $R_{t,r}$ is from the graph rules. Specifically, for step t, the reward $R_{t,f}$ contains both an intermediate reward and a final graph reward for graph G_{t+1} that

$$R_{t,f} = R_{t,f}(G_{t+1}) + \lambda_1 \frac{\sum_{i=1}^{m} R_{t,f}(\text{Rollout}(G_{t+1}))}{m}, \tag{5.10}$$

where λ_1 is a hyper-parameter, and the first term is the intermediate reward which can be obtained by feeding G_{t+1} to the trained GNNs $f(\cdot)$ and checking the predicted probability for class c_i. Mathematically, it can be computed as

$$R_{t,f}(G_{t+1}) = p(f(G_{t+1}) = c_i) - 1/\ell, \tag{5.11}$$

where ℓ denotes the number of possible classes for $f(\cdot)$. In addition, the second term in Equation (5.10) is the final graph reward for G_{t+1} which can be obtained by performing Rollout [57] m times on the intermediate graph G_{t+1}. Each time, a final graph is generated based on G_{t+1} until termination and then evaluated by $f(\cdot)$ using Equation (5.11). Then the evaluations for m final graphs are averaged to serve as the final graph reward. Overall, $R_{t,f}$ is positive when the obtained graph tends to yield high score for class c_i, and vice versa.

In addition, the reward $R_{t,r}$ is obtained from graphs rules and is employed to encourage the generated graphs to be valid and human-intelligible. The first rule we employ is that only one edge is allowed to be added between any two nodes. Second, the generated graph cannot contain more nodes than the predefined maximum node number. In addition, we incorporate dataset-specific rules to guide the graph generation. For example, in a chemical dataset, each node represents an atom so that its degree cannot exceed the valency of the corresponding atom. When any of these rules is violated, a negative reward will be applied for $R_{t,r}$. Finally, by combining the $R_{t,f}$ and

Algorithm 3 THE ALGORITHM OF OUR PROPOSED XGNN.

Given the trained GNNs for graph classification, denoted as $f(\cdot)$, we try to explain it and set the target class as c_i.
Let C define the candidate node set and $g(\cdot)$ mean our graph generator. We predefine the maximum generation step as S_{max} and the number of Rollout as m.
Define the initial graph as G_1.
for step t in S_{max} **do**
    Merge the current graph G_t and the candidate set C.
    Obtain the action a_t from the generator $g(\cdot)$ that $a_t = (a_{t,start}, a_{t,end})$ with Equation (5.4-5.8).
    Obtain the new graph G_{t+1} based on a_t.
    Evaluate G_{t+1} with Equation (5.10-5.12) and obtain R_t.
    Update the generator $g(\cdot)$ with Equation (5.9).
    if $R_t < 0$ **then** roll back and set $G_{t+1} = G_t$.
    end if
end for

$R_{t,r}$, we can obtain the reward for step t that

$$R_t = R_{t,f}(G_{t+1}) + \lambda_1 \frac{\sum_{i=1}^{m} R_{t,f}(\text{Rollout}(G_{t+1}))}{m} + \lambda_2 R_{t,r}, \tag{5.12}$$

where λ_1 and λ_2 are hyper-parameters. We illustrate the training procedure in Algorithm 3. Note that we roll back the graph G_{t+1} to G_t when the action a_t is evaluated as not promising that $R_t < 0$.

5.4 Experimental Studies

5.4.1 Dataset and Experimental Setup

We evaluate our proposed XGNN on both synthetic and real-world datasets. We report the summary statistics of these datasets in Table 5.1. Since there is no existing work investigating model-level explanations of GNNs, we have no baseline to compare with. Note that existing studies [88, 89] only focus on explaining GNNs at instance-level while ignoring the model-level explanations. Comparing with them is not expected since these instance-level and model-level are two totally different explanation directions.

Table 5.1: Statistics and properties of datasets. Note that the edge number and node number are averaged numbers.

Dataset	Classes	# of Edges	# of Nodes	Accuracy
Is_Acyclic	2	30.04	28.46	0.978
MUTAG	2	19.79	17.93	0.963

Synthetic dataset: Since our XGNN generates model-level explanations for Deep GNNs, we build a synthetic dataset, known as Is_Acyclic, where the ground truth explanations are available. The graphs are labeled based on if there is any cycle existing in the graph. The graphs are obtained using Networkx software package [101]. The first class refers to cyclic graphs, including grid-like graphs, cycle graphs, wheel graphs, and circular ladder graphs. The second class denotes acyclic graphs, containing star-like graphs, binary tree graphs, path graphs and full rary tree graphs [102]. Note that all nodes in this dataset are unlabeled and we focus on investigating the ability of GNNs to capture graph structures.

Real-world dataset: We conduct experiments on the real-world dataset MUTAG. The MUTAG dataset contains graphs representing chemical compounds where nodes represent different atoms and edges represent chemical bonds. The graphs are labeled into two different classes according to their mutagenic effect on a bacterium [103]. Each node is labeled based on its type of atom and there are seven possible atom types: Carbon, Nitrogen, Oxygen, Fluorine, Iodine, Chlorine, Bromine. Note that the edge labels are ignored for simplicity. For this dataset, we investigate the ability of GNNs to capture both graph structures and node labels.

Graph classification models: We train graph classification models using these datasets and then try to explain these models. These models share a similar pipeline that first learns node features using multiple layers of GCNs, then obtain graph level embeddings by averaging all node features, and finally employs fully-connected layers to perform graph classification. For the synthetic dataset Is_Acyclic, we use the node degrees as the initial features for all nodes. Then we apply two layers of GCNs with output dimensions equal to 8, 16 respectively and perform global averaging to obtain the graph representations. Finally, we employ one fully-connected layer as the classi-

fier. Meanwhile, for the real-world dataset MUTAG, since all nodes are labeled, we employ the corresponding one-hot representations as the initial node features. Then we employ three layers of GCNs with output dimensions equal to 32, 48, 64 respectively and average all node features. The final classifier contains two fully-connected layers in which the hidden dimension is set to 32. Note that for all GCN layers, we apply the GCN version shown in Equation (5.1). In addition, we employ Sigmoid as the non-linear function in GCNs for dataset Is_Acyclic while we use Relu for dataset MUTAG. These models are implemented using Pytorch [104] and trained using Adam optimizer [65]. The training accuracies of these models are reported in Table 5.1, which show that the models we try to explain are models with reasonable performance.

Graph generators: For both datasets, our graph generators share the same structure. Our generator first employs a fully-connected layer to map node features to the dimension of 8. Then three layers of GCNs are employed with output dimensions equal to 16, 24, 32 respectively. The first MLPs consist of two fully-connected layers with the hidden dimension equal to 16 and a ReLU6 non-linear function. The second MLPs also have two fully-connected layers that the hidden dimension is set to 24 and ReLU6 is applied. The initial features for input graphs are the same as mentioned above. For dataset Is_Acyclic, we set $\lambda_1 = 1$, $\lambda_2 = 1$, and $R_{t,r} = -1$ if the generated graph violates any graph rule. For dataset MUTAG, we set $\lambda_1 = 1$, $\lambda_2 = 2$, and the total reward $R_t = -1$ if the generated graph violates any graph rule. In addition, we perform rollout $m = 10$ times each step to obtain final graph rewards. The models are implemented using Pytorch [104] and trained using Adam optimizer [65] with $\beta_1 = 0.9$ and $\beta_2 = 0.999$. The learning rate for graph generator training is set to 0.01.

5.4.2 Experimental Results on Synthetic Data

We first conduct experiments on the synthetic dataset Is_Acyclic where the ground truth is available. As shown in Table 5.1, the trained GNN classifier can reach a promising performance. Since the dataset is manually and synthetically built based on if the graph contains any circle, we can check if the trained GNN classifier makes predictions in such a way. We explain the

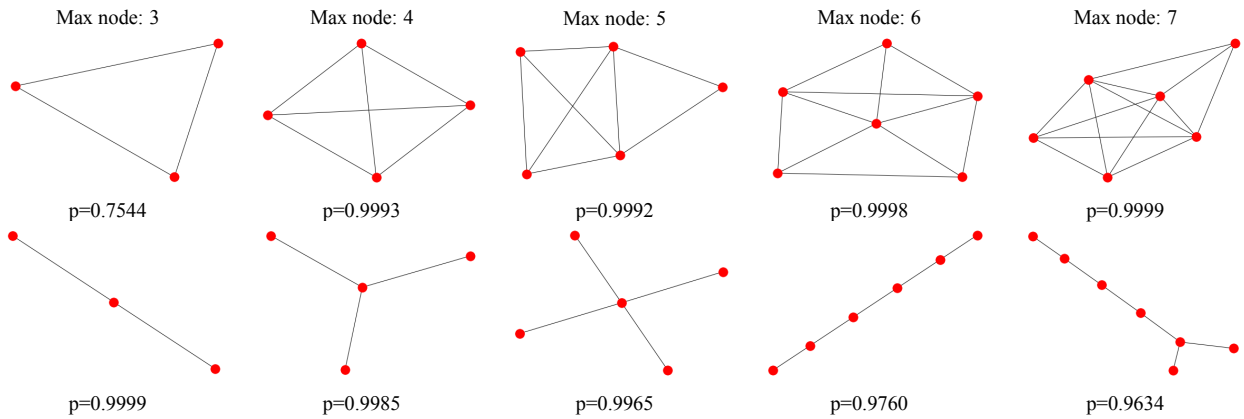

Figure 5.3: Experimental results for the synthetic dataset Is_Acyclic. Each row shows our explanations for a certain class that the first row corresponds to the class cyclic while the second row explains the class acyclic. In each row, from left to right, we report the generated graphs with increasing maximum node number limits. In addition, we feed each generated graph to the pre-trained GCNs and report the predicted probability for the corresponding class.

model with our proposed XGNN and report the generated explanations in Figure 5.3. We show the explanations for the class "cyclic" in the first row and the results for the class "acyclic" in the second row. In addition, we also report different generated explanations by setting different maximum graph node limits.

First, by comparing the graphs generated for different classes, we can easily conclude the difference that the explanations for the class "cyclic" always contain circles while the results for the class "acyclic" have no circle at all. Second, to verify whether our explanations can maximize the class probability for a certain class, as shown in Equation (5.2), we feed each generated graph to the trained GNN classifier and report the predicted probability for the corresponding class. The results show that our generated graph patterns can consistently yield high predicted probabilities. Note that even though the graph obtained for the class "cyclic" with maximum node number equal to 3 only leads to $p = 0.7544$, it is still the highest probability for all possible graphs with 3 nodes. Finally, based on these results, we can understand what patterns can maximize the predicted probabilities for different classes. In our results, we know the trained GNN classifier very likely distinguishes different classes by detecting circular structures, which is consistent with our

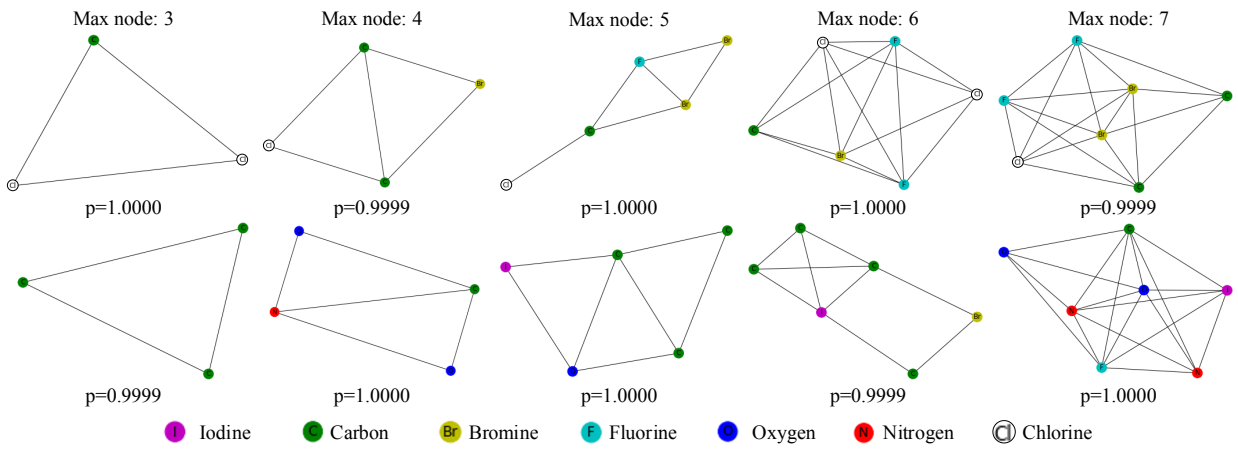

Figure 5.4: Experimental results for the MUTAG dataset. The first row reports the explanations for the class non-mutagenic while the second row shows results for the class mutagenic. Note that different node colors denote different types of atoms and the legend is shown at the bottom of the figure. All graphs are generated with the initial graph as a single Carbon atom.

expectations. Hence, such explanations help understand and trust the model, and increase the trustworthiness of this model to be used as a circular graph detector. In addition, it is noteworthy that our generated graphs are easier to analyze compared with the graphs in the datasets. Our generated graphs have significantly fewer numbers of nodes and simpler structures, and yield higher predicted probabilities while the graphs from the dataset have an average of 28 nodes and 30 edges, as shown in Table 5.1.

5.4.3 Experimental Results on Real-World Data

We also evaluate our proposed XGNN using real-world data. For dataset MUTAG, there is no ground truth for the explanations. Since all nodes are labeled as different types of atoms, we investigate whether the trained GNN classifier can capture both graph structures and node labels. We explain the trained GNN with our proposed method and report selected results in Figure 5.4 and Figure 5.5. Note that the generated graphs may not represent real chemical compounds because, for simplicity, we only incorporate a simple chemical rule that the degree of an atom cannot exceed its maximum chemical valency. In addition, since nodes are labeled, we can set the initial graphs as different types of atoms.

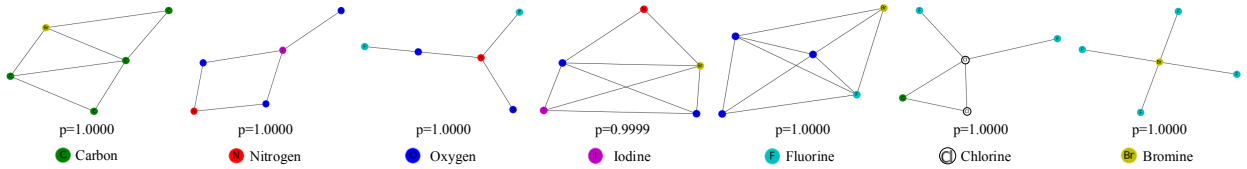

Figure 5.5: Experimental results for the MUTAG dataset. We fix the maximum node number limit as 5 and explore different initial graphs. Note that all graphs are generated for explaining the mutagenic class. For each generated graph, we show its predicted probability and corresponding initial graph at the bottom.

We first set the initial graph as a single carbon atom and report the results in Figure 5.4, since generally, any organic compound contains carbon [100]. The first row reports explanations for the class "non-mutagenic" while the second row shows the results for the class "mutagenic". We report the generated graphs with different node limits and the GNN predicted probabilities. For the class "mutagenic", we can observe that carbon circles and NO_2 are some common patterns, and this is consistent with the chemical fact that carbon rings and NO_2 chemical groups are mutagenic [103].

Such observations indicate that the trained GNN classifier may capture these key graph patterns to make predictions. In addition, for the class "non-mutagenic", we observe the atom Chlorine is widely existing in the generated graphs and the combination of Chlorine, Bromine, and Fluorine always leads to "non-mutagenic" predictions. By analyzing such explanations, we can better understand the trained GNN model.

We also explore different initial graphs and report the results in Figure 5.5. We fix the maximum node limit as 5 and generate explanations for the class "mutagenic". First, no matter how we set the initial graph, our proposed method can always find graph patterns maximizing the predicted probability of class "mutagenic". For the first 5 graphs, which means the initial graph is set to a single node of Carbon, Nitrogen, Oxygen, Iodine, or Fluorine, some generated graphs still have common patterns like carbon circle and NO_2 chemical groups. Our observations further confirm that these key patterns are captured by the trained GNNs. In addition, we notice that the generator can still produce graphs with Chlorine which are predicted as "mutagenic", which is contrary to our conclusion above. If all graphs with Chlorine should be identified as "non-mutagenic",

such explanations show the limitations of trained GNNs. Then these generated explanations can provide guidance for improving the trained GNNs, for example, we may place more emphasis on the graphs Chlorine when training the GNNs. Furthermore, the generated explanations may also be used to retrain and improve the GNN models to correctly capture our desired patterns. Overall, the experimental results show that our proposed explanation method XGNN can help verify, understand, and even help improve the trained GNN models.

6. CONCLUSIONS AND FUTURE WORK

In Chapter 2, we propose a learning-based method to generate discrete masks to explain deep image classifiers. In particular, we propose to explain deep models by learning the discriminative image regions in a GAN manner that we treat the pre-trained model as the discriminator and try to explain it using a trainable generator. The generator learns to capture important regions and produces a probability map. Then a discrete mask is sampled from this probability map and fed to the discriminator to measure its quality. We propose to train the generator using the policy gradient because of the sampling operations. Furthermore, to reduce the search space, we propose to incorporate auxiliary information. We conduct both quantitative and qualitative experiments to demonstrate the effectiveness of our proposed method. The visual results show that our method obtains better explanations than several state-of-the-art approaches. In addition, our proposed method can pass the model randomization test, showing that our method is reasoning the predictions instead of guessing. The quantitative analysis via weakly supervised localization task and saliency metric demonstrates the effectiveness of our proposed method. We also perform the ROAR evaluation for our method and further show that our method can correctly identify important and discriminative image regions for the predictions of models. Finally, the ablation study demonstrates that both the area reward and smoothness reward are important to generate good explanations.

In Chapter 3, we propose to study the meaning of the neurons in the hidden layers, thereby explaining the whole prediction procedures layer by layer. While investigating hidden units in neural networks are of great importance to understand their working mechanisms, it is challenging to understand the meaning of hidden units in NLP models, since word representations are discrete and cannot be abstracted. Our proposed method first employs gradient-based approaches to estimate the contributions of different spatial locations in a hidden layer and then uses optimization to answer the question of what is detected by these hidden locations. Then we propose to approximately explain the meaning of detected information using the nearest neighbors of the optimized representation based on the special property of word representations that words with semantically

similar meanings are embedded to nearby points. Experimental results show that our approaches can identify reasonable explanations for hidden locations, which shares similar high-level meaning with the input sentence. It is also shown that our method helps explain how the decision and why the decision is made.

In Chapter 4, we propose a novel instance-level explanation method, known as SubgraphX, to study graph neural networks. While considerable efforts have been devoted to study the explainability of GNNs, none of the existing methods can explain GNN predictions with subgraphs. We argue that subgraphs are building blocks of complex graphs and are more human-intelligible. To this end, we propose the SubgraphX to explain GNNs by identifying important subgraphs explicitly. We employ the Monte Carlo tree search algorithm to efficiently explore different subgraphs. For each subgraph, we propose to employ Shapley values to measure its importance by considering the interactions among different graph structures. To expedite computations, we propose efficient approximation schemes to compute Shapley values by considering interactions only within the information aggregation range. Experimental results show our SubgraphX obtain higher-quality and more human-intelligible explanations while keeping time complexity acceptable.

In Chapter 5, we propose the XGNN to provide model-level explanations for graph neural networks. The core idea of our general XGNN framework is to find graph patterns that can maximize a certain prediction via graph generation. Specifically, we formulate it as a reinforcement learning problem and generate graph patterns iteratively. We train a graph generator and for each step, it predicts how to add an edge into the current graph. In addition, we incorporate several graph rules to encourage the generated graphs to be valid and human-intelligible. We conduct thorough experiments on both synthetic and real-world datasets to demonstrate the effectiveness of our proposed XGNN. The results show that the generated graphs help discover what patterns will maximize a certain prediction of the trained GNNs. The generated explanations help verify and better understand if the trained GNNs make predictions in our expected way. Furthermore, our results also show that the generated explanations can provide directions to improve the trained models.

In terms of future work, we discuss several possible directions for future work. First, in our

proposed SubgraphX, we employ the search algorithm for exploring subgraphs and compute the Shapley value for each subgraph individually. Hence, it is important to further improve the efficiency, and possible solutions include designing proper greedy algorithms to reduce the time cost for searching and computing the Shapley values for multiple subgraphs simultaneously based on the addictive property of Shapley values. Second, while we investigate the explainability of deep models for different domains, the target models are relatively simple. The explainability of more complex models, such as Transformer [6], BERT [141], and Recommendation Systems [142], has not been explored yet. Explaining such complex models is an important but more challenging task since these complex models are widely used in real-world applications. Last but not the least, explaining critical applications of deep models is also important. One example is the deep learning model for medical images. Since medical decisions are critical, it is not enough to only make predictions based on medical images. It is also necessary to provide explanations of why the predictions are made. Otherwise, the doctors or patients cannot trust the models at all. Since medical images are different from natural images, existing models cannot be directly applied. Hence, specific techniques are needed to explain the deep models trained with medical data.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[2] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 4, pp. 834–848, 2018.

[3] C. Doersch, "Tutorial on variational autoencoders," *arXiv preprint arXiv:1606.05908*, 2016.

[4] S. Ji, W. Xu, M. Yang, and K. Yu, "3d convolutional neural networks for human action recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 1, pp. 221–231, 2013.

[5] Y. Kim, "Convolutional neural networks for sentence classification," *arXiv preprint arXiv:1408.5882*, 2014.

[6] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.

[7] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," in *Advances in neural information processing systems*, pp. 649–657, 2015.

[8] H. Yuan and S. Ji, "Structpool: Structured graph pooling via conditional random fields," in *International Conference on Learning Representations*, 2020.

[9] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. LiÃš, and Y. Bengio, "Graph attention networks," in *International Conference on Learning Representations*, 2018.

[10] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[11] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[12] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proceedings of the International Conference on Learning Representations*, 2017.

[13] B. Goodman and S. Flaxman, "European union regulations on algorithmic decision-making and a âĂIJright to explanationâĂİ," *AI magazine*, vol. 38, no. 3, pp. 50–57, 2017.

[14] D. Erhan, Y. Bengio, A. Courville, and P. Vincent, "Visualizing higher-layer features of a deep network," *University of Montreal*, vol. 1341, no. 3, p. 1, 2009.

[15] A. Nguyen, J. Clune, Y. Bengio, A. Dosovitskiy, and J. Yosinski, "Plug & play generative networks: Conditional iterative generation of images in latent space," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3510–3520, IEEE, 2017.

[16] A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 427–436, 2015.

[17] H. Yuan, J. Tang, X. Hu, and S. Ji, "XGNN: Towards model-level explanations of graph neural networks," KDD '20, (New York, NY, USA), p. 430âĂŞ438, Association for Computing Machinery, 2020.

[18] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," *arXiv preprint arXiv:1312.6034*, 2013.

[19] D. Smilkov, N. Thorat, B. Kim, F. Viégas, and M. Wattenberg, "Smoothgrad: removing noise by adding noise," *arXiv preprint arXiv:1706.03825*, 2017.

[20] 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," in *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 618–626, IEEE, 2017.

[21] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2921–2929, 2016.

[22] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*, pp. 818–833, Springer, 2014.

[23] R. C. Fong and A. Vedaldi, "Interpretable explanations of black boxes by meaningful perturbation," in *2017 IEEE international conference on computer vision (ICCV)*, pp. 3449–3457, IEEE, 2017.

[24] P. Dabkowski and Y. Gal, "Real time image saliency for black box classifiers," in *Advances in Neural Information Processing Systems*, pp. 6967–6976, 2017.

[25] C. Olah, A. Mordvintsev, and L. Schubert, "Feature visualization," *Distill*, 2017. https://distill.pub/2017/feature-visualization.

[26] J. Zhang, S. A. Bargal, Z. Lin, J. Brandt, X. Shen, and S. Sclaroff, "Top-down neural attention by excitation backprop," *International Journal of Computer Vision*, vol. 126, no. 10, pp. 1084–1102, 2018.

[27] S. Bach, A. Binder, G. Montavon, F. Klauschen, K. Müller, and W. Samek, "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation," *PLoS ONE*, vol. 10, 2015.

[28] G. Montavon, S. Lapuschkin, A. Binder, W. Samek, and K. Müller, "Explaining nonlinear classification decisions with deep taylor decomposition," *Pattern Recognit.*, vol. 65, pp. 211–222, 2017.

[29] M. T. Ribeiro, S. Singh, and C. 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.

[30] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Advances in neural information processing systems*, pp. 4765–4774, 2017.

[31] C. Olah, A. Satyanarayan, I. Johnson, S. Carter, L. Schubert, K. Ye, and A. Mordvintsev, "The building blocks of interpretability," *Distill*, 2018. https://distill.pub/2018/building-blocks.

[32] J. Li, X. Chen, E. Hovy, and D. Jurafsky, "Visualizing and understanding neural models in nlp," *arXiv preprint arXiv:1506.01066*, 2015.

[33] H. Yuan, Y. Chen, X. Hu, and S. Ji, "Interpreting deep models for text analysis via optimization and regularization methods," in *Thirty-Third AAAI Conference on Artificial Intelligence*, pp. 5717–5724, 2019.

[34] J. Chen, L. Song, M. J. Wainwright, and M. I. Jordan, "Learning to explain: An information-theoretic perspective on model interpretation," in *International Conference on Machine Learning*, vol. 80 of *Proceedings of Machine Learning Research*, pp. 882–891, PMLR, 2018.

[35] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2015.

[36] G. Huang, Z. Liu, and K. Q. Weinberger, "Densely connected convolutional networks," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2261–2269, 2017.

[37] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241, Springer, 2015.

[38] A. Fakhry, T. Zeng, and S. Ji, "Residual deconvolutional networks for brain electron microscopy image segmentation," *IEEE transactions on medical imaging*, vol. 36, no. 2, pp. 447–456, 2017.

[39] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 12, pp. 2481–2495, 2017.

[40] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, pp. 2672–2680, 2014.

[41] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *CoRR*, vol. abs/1312.6114, 2014.

[42] J. Adebayo, J. Gilmer, M. Muelly, I. Goodfellow, M. Hardt, and B. Kim, "Sanity checks for saliency maps," in *Advances in Neural Information Processing Systems*, pp. 9525–9536, 2018.

[43] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net," *International Conference on Learning Representations*, 2015.

[44] M. Du, N. Liu, Q. Song, and X. Hu, "Towards explanation of dnn-based prediction with guided feature inversion," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1358–1367, 2018.

[45] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014.

[46] M. Du, N. Liu, and X. Hu, "Techniques for interpretable machine learning," *Communications of the ACM*, vol. 63, no. 1, pp. 68–77, 2019.

[47] D. Baehrens, T. Schroeter, S. Harmeling, M. Kawanabe, K. Hansen, and K.-R. MÃžller, "How to explain individual classification decisions," *Journal of Machine Learning Research*, vol. 11, no. Jun, pp. 1803–1831, 2010.

[48] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks," in *International Conference on Machine Learning*, pp. 3319–3328, 2017.

[49] A. Shrikumar, P. Greenside, and A. Kundaje, "Learning important features through propagating activation differences," in *International Conference on Machine Learning*, pp. 3145–3153, 2017.

[50] A. Mahendran and A. Vedaldi, "Understanding deep image representations by inverting them," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5188–5196, 2015.

[51] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," in *International conference on machine learning*, pp. 2048–2057, 2015.

[52] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," in *International Conference on Learning Representations*, 2016.

[53] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations*, 2015.

[54] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9, June 2015.

[55] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[56] T. Lei, R. Barzilay, and T. Jaakkola, "Rationalizing neural predictions," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 107–117, 2016.

[57] L. Yu, W. Zhang, J. Wang, and Y. Yu, "Seqgan: sequence generative adversarial nets with policy gradient," in *AAAI-17: Thirty-First AAAI Conference on Artificial Intelligence*, vol. 31, pp. 2852–2858, Association for the Advancement of Artificial Intelligence, 2017.

[58] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems*, pp. 1057–1063, 2000.

[59] H. Gao, H. Yuan, Z. Wang, and S. Ji, "Pixel transposed convolutional networks," *IEEE transactions on pattern analysis and machine intelligence*, vol. 42, no. 5, pp. 1218–1227, 2019.

[60] H. Yuan, L. Cai, Z. Wang, X. Hu, S. Zhang, and S. Ji, "Computational modeling of cellular structures using conditional deep generative networks," *Bioinformatics*, vol. 35, no. 12, pp. 2141–2149, 2019.

[61] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[62] N. Silberman and S. Guadarrama, "Tensorflowslim image classification model library," 2017.

[63] S. Ioffe and C. Szegedy, "Batch normalization: accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning-Volume 37*, pp. 448–456, JMLR. org, 2015.

[64] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: a system for large-scale machine learning," in *Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation*, pp. 265–283, USENIX Association, 2016.

[65] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proceedings of the 3rd International Conference on Learning Representations*, 2014.

[66] S. Bach, A. Binder, G. Montavon, F. Klauschen, K. Müller, and W. Samek, "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation," *PLoS ONE*, vol. 10, no. 7, p. e0130140, 2015.

[67] S. Hooker, D. Erhan, P.-J. Kindermans, and B. Kim, "A benchmark for interpretability methods in deep neural networks," in *Advances in Neural Information Processing Systems*, pp. 9734–9745, 2019.

[68] K. Lin, D. Li, X. He, Z. Zhang, and M.-T. Sun, "Adversarial ranking for language generation," in *Advances in Neural Information Processing Systems*, pp. 3155–3165, 2017.

[69] J. Gehring, M. Auli, D. Grangier, and Y. N. Dauphin, "A convolutional encoder model for neural machine translation," *arXiv preprint arXiv:1611.02344*, 2016.

[70] Z. Wang and S. Ji, "Learning convolutional text representations for visual question answering," in *Proceedings of the 2018 SIAM International Conference on Data Mining*, pp. 594–602, SIAM, 2018.

[71] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[72] A. Mordvintsev, C. Olah, and M. Tyka, "Inceptionism: Going deeper into neural networks," *Google Research Blog. Retrieved June*, vol. 20, no. 14, p. 5, 2015.

[73] A. Nguyen, J. Yosinski, Y. Bengio, A. Dosovitskiy, and J. Clune, "Plug & play generative networks: Conditional iterative generation of images in latent space," *arXiv preprint arXiv:1612.00005*, 2016.

[74] N. S. Altman, "An introduction to kernel and nearest-neighbor nonparametric regression," *The American Statistician*, vol. 46, no. 3, pp. 175–185, 1992.

[75] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[76] S. Wold, K. Esbensen, and P. Geladi, "Principal component analysis," *Chemometrics and intelligent laboratory systems*, vol. 2, no. 1-3, pp. 37–52, 1987.

[77] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010.

[78] H. Gao and S. Ji, "Graph U-Net," in *International conference on machine learning*, pp. 2083–2092, 2019.

[79] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?," in *International Conference on Learning Representations*, 2019.

[80] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *Thirty-Second AAAI Conference on Artificial Intelligence*, pp. 4438–4445, 2018.

[81] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," in *Advances in Neural Information Processing Systems*, pp. 5165–5175, 2018.

[82] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1263–1272, JMLR. org, 2017.

[83] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in neural information processing systems*, pp. 1024–1034, 2017.

[84] J. Lee, I. Lee, and J. Kang, "Self-attention graph pooling," in *International Conference on Machine Learning*, pp. 3734–3743, 2019.

[85] K. K. Thekumparampil, C. Wang, S. Oh, and L.-J. Li, "Attention-based graph neural network for semi-supervised learning," *arXiv preprint arXiv:1803.03735*, 2018.

[86] H. Gao and S. Ji, "Graph representation learning via hard and channel-wise attention networks," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 741–749, 2019.

[87] F. Doshi-Velez and B. Kim, "Towards a rigorous science of interpretable machine learning," *arXiv preprint arXiv:1702.08608*, 2017.

[88] Z. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec, "Gnnexplainer: Generating explanations for graph neural networks," in *Advances in Neural Information Processing Systems*, pp. 9244–9255, 2019.

[89] F. Baldassarre and H. Azizpour, "Explainability techniques for graph convolutional networks," in *International Conference on Machine Learning (ICML) Workshops, 2019 Workshop on Learning and Reasoning with Graph-Structured Representations*, 2019.

[90] M. Gevrey, I. Dimopoulos, and S. Lek, "Review and comparison of methods to study the contribution of variables in artificial neural network models," *Ecological modelling*, vol. 160, no. 3, pp. 249–264, 2003.

[91] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon, "Network motifs: simple building blocks of complex networks," *Science*, vol. 298, no. 5594, pp. 824–827, 2002.

[92] U. Alon, *An introduction to systems biology: design principles of biological circuits*. Chapman and Hall/CRC, 2006.

[93] U. Alon, "Network motifs: theory and experimental approaches," *Nature Reviews Genetics*, vol. 8, no. 6, p. 450, 2007.

[94] S. S. Shen-Orr, R. Milo, S. Mangan, and U. Alon, "Network motifs in the transcriptional regulation network of escherichia coli," *Nature genetics*, vol. 31, no. 1, p. 64, 2002.

[95] H. Wang, J. Wang, J. Wang, M. Zhao, W. Zhang, F. Zhang, X. Xie, and M. Guo, "GraphGAN: Graph representation learning with generative adversarial nets," in *Thirty-Second AAAI Conference on Artificial Intelligence*, pp. 2508–2515, 2018.

[96] G. L. Guimaraes, B. Sanchez-Lengeling, C. Outeiral, P. L. C. Farias, and A. Aspuru-Guzik, "Objective-reinforced generative adversarial networks (ORGAN) for sequence generation models," *arXiv preprint arXiv:1705.10843*, 2017.

[97] W. Jin, R. Barzilay, and T. Jaakkola, "Junction tree variational autoencoder for molecular graph generation," in *Proceedings of the 35th International Conference on Machine Learning*, pp. 2323–2332, 2018.

[98] Y. Li, O. Vinyals, C. Dyer, R. Pascanu, and P. Battaglia, "Learning deep generative models of graphs," *arXiv preprint arXiv:1803.03324*, 2018.

[99] J. You, B. Liu, Z. Ying, V. Pande, and J. Leskovec, "Graph convolutional policy network for goal-directed molecular graph generation," in *Advances in Neural Information Processing Systems*, pp. 6410–6421, 2018.

[100] S. L. Seager and M. R. Slabaugh, *Chemistry for today: General, organic, and biochemistry*. Cengage learning, 2013.

[101] A. Hagberg, P. Swart, and D. S Chult, "Exploring network structure, dynamics, and function using networkx," tech. rep., Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[102] J. A. Storer, *An introduction to data structures and algorithms*. Springer Science & Business Media, 2012.

[103] A. K. Debnath, R. L. Lopez de Compadre, G. Debnath, A. J. Shusterman, and C. Hansch, "Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity," *Journal of medicinal chemistry*, vol. 34, no. 2, pp. 786–797, 1991.

[104] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *Proceedings of the International Conference on Learning Representations*, 2017.

[105] H. Gao, Z. Wang, and S. Ji, "Large-scale learnable graph convolutional networks," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1416–1424, 2018.

[106] Z. Wang, M. Liu, Y. Luo, Z. Xu, Y. Xie, L. Wang, L. Cai, and S. Ji, "Advanced graph and sequence neural networks for molecular property prediction and drug discovery," *arXiv preprint arXiv:2012.01981*, 2020.

[107] X. Wang, H. Ji, C. Shi, B. Wang, Y. Ye, P. Cui, and P. S. Yu, "Heterogeneous graph attention network," in *The World Wide Web Conference*, pp. 2022–2032, 2019.

[108] H. Yuan, L. Cai, X. Hu, J. Wang, and S. Ji, "Interpreting image classifiers by generating discrete masks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[109] F. Yang, S. K. Pentyala, S. Mohseni, M. Du, H. Yuan, R. Linder, E. D. Ragan, S. Ji, and X. Hu, "Xfake: explainable fake news detector with visualizations," in *The World Wide Web Conference*, pp. 3600–3604, 2019.

[110] Z. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec, "Gnnexplainer: Generating explanations for graph neural networks," in *Advances in neural information processing systems*, pp. 9244–9255, 2019.

[111] D. Luo, W. Cheng, D. Xu, W. Yu, B. Zong, H. Chen, and X. Zhang, "Parameterized explainer for graph neural network," in *Advances in neural information processing systems*, 2020.

[112] M. N. Vu and M. T. Thai, "Pgm-explainer: Probabilistic graphical model explanations for graph neural networks," in *Advances in neural information processing systems*, 2020.

[113] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, "Mastering the game of go without human knowledge," *nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[114] H. W. Kuhn and A. W. Tucker, *Contributions to the Theory of Games*, vol. 2. Princeton University Press, 1953.

[115] P. E. Pope, S. Kolouri, M. Rostami, C. E. Martin, and H. Hoffmann, "Explainability methods for graph convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 10772–10781, 2019.

[116] M. S. Schlichtkrull, N. De Cao, and I. Titov, "Interpreting graph neural networks for nlp with differentiable edge masking," *arXiv preprint arXiv:2010.00577*, 2020.

[117] C. Louizos, M. Welling, and D. P. Kingma, "Learning sparse neural networks through l_0 regularization," *arXiv preprint arXiv:1712.01312*, 2017.

[118] Anonymous, "Hard masking for explaining graph neural networks," in *Submitted to International Conference on Learning Representations*, 2021. under review.

[119] Anonymous, "Causal screening to interpret graph neural networks," in *Submitted to International Conference on Learning Representations*, 2021. under review.

[120] Q. Huang, M. Yamada, Y. Tian, D. Singh, D. Yin, and Y. Chang, "Graphlime: Local interpretable model explanations for graph neural networks," *arXiv preprint arXiv:2001.06216*, 2020.

[121] M. T. Ribeiro, S. Singh, and C. Guestrin, ""why should i trust you?" explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1135–1144, 2016.

[122] M. Yamada, W. Jitkrittum, L. Sigal, E. P. Xing, and M. Sugiyama, "High-dimensional feature selection by feature-wise kernelized lasso," *Neural computation*, vol. 26, no. 1, pp. 185–207, 2014.

[123] Y. Zhang, D. Defazio, and A. Ramesh, "Relex: A model-agnostic relational model explainer," *arXiv preprint arXiv:2006.00305*, 2020.

[124] D. Margaritis and S. Thrun, "Bayesian network induction via local neighborhoods," *Advances in neural information processing systems*, vol. 12, pp. 505–511, 1999.

[125] R. Schwarzenberg, M. Hübner, D. Harbecke, C. Alt, and L. Hennig, "Layerwise relevance visualization in convolutional text graph classifiers," *arXiv preprint arXiv:1909.10911*, 2019.

[126] T. Schnake, O. Eberle, J. Lederer, S. Nakajima, K. T. SchÃijtt, K.-R. MÃijller, and G. Montavon, "Higher-order explanations of graph neural networks via relevant walks," 2020.

[127] U. Alon, *An introduction to systems biology: design principles of biological circuits*. CRC press, 2019.

[128] H. Yuan, H. Yu, S. Gui, and S. Ji, "Explainability in graph neural networks: A taxonomic survey," *arXiv preprint arXiv:2012.15445*, 2020.

[129] W. Jin, R. Barzilay, and T. Jaakkola, "Multi-objective molecule generation using interpretable substructures," in *International Conference on Machine Learning*, pp. 4849–4859, PMLR, 2020.

[130] J. Chen, L. Song, M. J. Wainwright, and M. I. Jordan, "L-shapley and c-shapley: Efficient model interpretation for structured data," *arXiv preprint arXiv:1808.02610*, 2018.

[131] E. Štrumbelj and I. Kononenko, "Explaining prediction models and individual predictions with feature contributions," *Knowledge and information systems*, vol. 41, no. 3, pp. 647–665, 2014.

[132] M. S. Schlichtkrull, N. D. Cao, and I. Titov, "Interpreting graph neural networks for {nlp} with differentiable edge masking," in *International Conference on Learning Representations*, 2021.

[133] Z. Chen, X. Li, and J. Bruna, "Supervised community detection with line graph neural networks," *arXiv preprint arXiv:1705.08415*, 2017.

[134] A. Jacovi and Y. Goldberg, "Towards faithfully interpretable nlp systems: How should we define and evaluate faithfulness?," *arXiv preprint arXiv:2004.03685*, 2020.

[135] S. Wiegreffe and Y. Pinter, "Attention is not not explanation," *arXiv preprint arXiv:1908.04626*, 2019.

[136] S. Hooker, D. Erhan, P.-J. Kindermans, and B. Kim, "A benchmark for interpretability methods in deep neural networks," in *Advances in Neural Information Processing Systems*, pp. 9737–9748, 2019.

[137] B. Sanchez-Lengeling, J. Wei, B. Lee, E. Reif, P. Wang, W. W. Qian, K. McCloskey, L. Colwell, and A. Wiltschko, "Evaluating attribution for graph neural networks," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[138] A. K. Debnath, R. L. Lopez de Compadre, G. Debnath, A. J. Shusterman, and C. Hansch, "Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity," *Journal of medicinal chemistry*, vol. 34, no. 2, pp. 786–797, 1991.

[139] Z. Wu, B. Ramsundar, E. N. Feinberg, J. Gomes, C. Geniesse, A. S. Pappu, K. Leswing, and V. Pande, "Moleculenet: a benchmark for molecular machine learning," *Chemical science*, vol. 9, no. 2, pp. 513–530, 2018.

[140] M. Gardner, J. Grus, M. Neumann, O. Tafjord, P. Dasigi, N. Liu, M. Peters, M. Schmitz, and L. Zettlemoyer, "Allennlp: A deep semantic natural language processing platform," *arXiv preprint arXiv:1803.07640*, 2018.

[141] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[142] M. J. Pazzani and D. Billsus, "Content-based recommendation systems," in *The adaptive web*, pp. 325–341, Springer, 2007.