Leibniz Universität Hannover

Fakultät für Elektrotechnik und Informatik
Institut für Verteilte Systeme
Fachgebiet Wissensbasierte Systeme

Explaining Graph Neural Networks

Bachelor thesis

In partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science.

Author:	Tobias Christian Nauen
Matriculation Number:	10006435
First Examiner:	Prof. Dr. Avishek Anand
Second Examiner & Supervisor:	Dr. Thorben Funke
Date:	October 8, 2021

Abstract

Graph Neural Networks are an up-and-coming class of neural networks that operate on graphs and can therefore deal with connected, highly complex data. As explaining neural networks becomes more and more important, we investigate different ways to explain graph neural networks and contrast gradient-based explanations with the interpretability by design approach KEdge. We extend KEdge, to work with probability distributions different from HardKuma. Our goal is to test the performance of each method to judge which one works best under given circumstances. For this, we extend the notion of fidelity from hard attribution weights to soft attribution weights and use the resulting metric to evaluate the explanations generated by KEdge, as well as by the gradient-based techniques. We also compare the predictive performance of models that use KEdge with different distributions. Our experiments are run on the Cora, SightSeer, Pubmed, and MUTAG datasets. We find that KEdge outperforms the gradient based attribution techniques on graph classification problems and that it should be used with the HardNormal, HardKuma, or HardLaplace distributions, depending on if the top priority is model performance or attribution quality. To compare different metrics of judging attributions in the text domain, we visualize attribution weights generated by different models, and find, that metrics which compare model attributions to human explanations lead to bad attribution weights.

Contents

Notation and Abbreviations

GNN	Graph neural network.	
G	Graph.	Definition 2.1.1
V	Set of vertices of a graph.	Definition 2.1.1
E	Set of edges of a graph.	Definition 2.1.1
A	Adjacency matrix of a graph *or* a matrix of attribution weights.	Definition 2.1.1
$\mathbb{1}_\Omega$	Indicator function on some set Ω.	
$\text{In}(v)$	Set of incident edges to the node $v \in V$.	Definition 2.1.1
$\mathcal{N}(v)$	Set of neighbors of the node $v \in V$.	Definition 2.1.1
X	Matrix of node features with rows x_v for $v \in V$.	
$H^{(k)}; h_v^{(k)}$	Matrix / vector of the k-th node representations. $H^{(0)} = X$.	Page 4
I	Identity matrix.	
$\|\|$	Concatenation operator.	
GAP	Global average pooling.	Section 2.1.5
GAT	Global attention pooling.	Section 2.1.5
NN	A neural network.	
$\|\cdot\|_2$	Euclidean norm.	
$\langle\cdot,\cdot\rangle$	Standard scalar product.	
$\odot$	Pointwise multiplication operator.	
Z	Matrix of mask weights.	
λ	Maskedness parameter *or* a real parameter.	Equation (3)
$\mathbb{R}$	Set of real numbers.	
$\mathcal{B}(\mathbb{R})$	Borel sigma algebra on $\mathbb{R}$.	
δ_x	Point measure in x; $\delta_x(A) := \mathbb{1}_A(x)$.	
$\mathbb{R}^+$	Set of positive real numbers.	
$fid(A)$	Fidelity of the hard attribution matrix A.	Definition 3.3.1
comp	Comprehensiveness.	Definition 3.3.4
suff	Sufficiency.	Definition 3.3.4
$AOPC$	Area over the perturbation curve.	Definition 3.3.5
$AUPRC$	Area under the precision-recall-curve.	Definition 3.3.8

1 Introduction

In this bachelor thesis, we explore and evaluate different methods of explaining Graph Neural Networks (GNNs). Graph Neural Networks are an emerging class of neural networks, that take graphs as their input data. This is especially useful since graphs are highly flexible and powerful data structures, that can therefore express a set of different datapoints with complex relationships between them. The motivation for developing graph neural networks comes from the overwhelming success of convolutional neural networks, which can be seen as a special case of GNNs, operating on pictures by exploiting neighborhood information, which can also be expressed as a graph. Today graph neural networks are used in a wide array of domains, like the prediction of molecular properties in chemistry [SK20], drug discovery [BBS21], or even diagnosis [BMR21] in medicine, to model the spread of disease [Kap+20], in recommendation systems [Yin+18], or natural language processing [Pen+18].
But why would one want to explain these networks? Methods for explaining neural models are used to perform a wide amount of tasks. The first one is to debug the model and increase performance, as explanation methods can uncover model bias or spurious correlations in the training data. These are then used, to clean up or expand the training data or to adjust the model class, to archive better performance and generalization.

Model	Prediction	Explanation
A	Positive	Even though the Icelandic scenery is incredibly stunning, the story can't keep up, and therefore the overall experience is boring.
B	Negative	Even though the Icelandic scenery is incredibly stunning, the story can't keep up, and therefore the overall experience is boring.
C	Negative	Even though the Icelandic scenery is incredibly stunning, the story can't keep up, and therefore the overall experience is boring.

Table 1: Hypothetical movie review with classifications and explanations.

For example, if we want to categorize reviews into positive and negative ones, we are also interested in exactly why our model decides that a given review is positive or negative. Using this information we can more accurately judge the models' performance, by checking if its predictions are correct for the right reasons. The example explanations in Table 1 reveal that model B is correct for the wrong reason, while model C is correct for the right reason. Therefore model C should be deployed over model B since we can expect C to generalize better to new, unseen data.
A second application area of explanations is to assess the suitability of a model for use in the real world. This is especially important in high-stakes environments, such as medicine or law enforcement, where graph neural networks are used. Therefore, explainability is also part of the approval process by a regulatory authority, like the European Union [Uni16; GF17], or in some companies. Another way explanation techniques are useful is by hinting on what to change in the input, to receive a different model output. This is useful for example in loan approval if the client wants to receive information on what factors to change to be approved. [LAS20]
One distinguishes two forms of explanations: global and local ones. While global explanations are ways of explaining the model as a whole, it is often not feasible to construct such global explanations, especially when using a model with a lot of parameters, since it's just too complex to be understood as a whole [LAS20]. Therefore, we want to focus on local explanations. These don't attempt to explain the whole model, but just a single decision of the model given a certain input. The explanations in Table 1 are local ones.

Now it begs the question, how one can explain the decisions of a graph neural network. To answer this question we will lay out the relevant techniques to generate attribution weights, as well as expand on them. Attribution weights are ways of explaining neural models by associating a weight with different parts, or tokens, of the models' input. These tokens could be pixels in a picture, words in a text, or nodes and/or edges in a graph. The parts with high weight are seen as more important for the models' decision than those with low weights. If all generated weights are zero or one, the technique is called a hard attribution technique. These mark relevant parts of the input, as is the case in Table 1. When a range of real numbers is allowed as weights, the attributions are called soft. We will focus on soft attribution techniques, as these provide a relation of importance on the inputs' tokens.
The second question that arises is, which technique should one use to explain GNNs and how to judge if one technique is better than another? To answer these questions, we first establish and explain the notion of graph neural networks, as well as different architectures (Section 2.1). Then we introduce some gradient-based attribution techniques (Section 2.2) and the interpretability by design approach of KEdge (Section 2.3). KEdge was introduced by Rathee et al. in 2021. It works by sampling a mask for the edges of a graph via an approximation of the Bernoulli distribution. This mask can then be used to generate attribution weights. In the original paper, this approximation is based on the Kumaraswamy, or Kuma, distribution. In our third chapter, we define some probability distributions (Section 3.1) to construct different approximations of the Bernoulli distribution, that we can use with KEdge. We also talk about how to obtain node-level attribution weights from KEdge (Section 3.2). Then we introduce some metrics to measure the performance of the different attribution techniques (Section 3.3), and in particular, we extend the notion of fidelity [FKA21] to soft attribution techniques by introducing integrated fidelity.
In the main part of this thesis, we conduct three experiments. The first two, to evaluate and compare the attribution techniques, as well as to see, what effects KEdge has on a model's performance. Here, we compare the accuracy of different models with and without KEdge, to see if there is a noticeable difference, depending on which underlying probability distribution we used. We also compare the integrated fidelity values of all the attribution techniques we introduced before. This is done on the node classification datasets Pubmed, Cora, and CiteSeer and the graph classification dataset MUTAG. In the last experiment, we use our methods on a text dataset of movie reviews, to be able to visualize attribution weights and compare different metrics of evaluating attribution weights.

Contributions

In this thesis, we contribute the following, new ideas:

- Implementation and evaluation of KEdge with distributions other than Kuma and Hard-Kuma,
- Extension of fidelity to soft masks (integrated fidelity),
- Comparison of different distributions for KEdge based on model performance on multiple datasets,
- Comparison of KEdge to gradient-based attribution techniques based on integrated fidelity on multiple datasets,
- Comparison of different metrics for the evaluation of attributions, when using GNNs in the text domain.

2 Background

2.1 Graph Neural Networks

As a first step to understanding and explaining Graph Neural Networks (GNNs), one needs to know what they are and how they work. As the name suggests, Graph Neural Networks are a family of neural models operating on graph data. To explain this further, let us first define graphs and some related notions.

Definition 2.1.1 (Graph)**:**
A *graph* is a tuple $G = (V, E)$. V is the set of *vertices* or *nodes* and $E \subset V \times V$ is the set of *edges*.
We call two edges v and v' *neighbors*, if $(v, v') \in E$ or $(v', v) \in E$.
The graph $G = (V, E)$ is called *finite*, if $n := |V| < \infty$. Henceforth, all graphs will be finite. For a finite graph we define the *adjacency matrix* A by letting $V = \{v_1, ..., v_n\}$ and

$$\mathbb{R}^{n \times n} \ni A := (a_{ij})_{ij} := (\mathbb{1}_{(v_i, v_j) \in E})_{ij}.$$

The adjacency matrix A encodes the edges of G and $a_{ij} = 1$ precisely, when there is an edge from node v_i to node v_j. We define the *degree* of a node $v \in V$ as the number of incoming and outgoing edges of v

$$\deg v := |(\{v\} \times V) \cap E| + |(V \times \{v\}) \cap E|$$

and the *degree matrix* by

$$\mathbb{R}^{n \times n} \ni D := (\delta_{ij} \deg v_i)_{ij},$$

where δ_{ij} denotes the Kronecker delta. D is the diagonal matrix of the node degrees.
Additionally, we define the set of *incident edges* of the node $v \in V$ by

$$\mathrm{In}(v) := \{(u, w) \in E | u = v \text{ or } w = v\} \subset E,$$

and its *neighborhood* by

$$\mathcal{N}(v) := \{u \in V | (u, v) \in E \text{ or } (v, u) \in E\} \subset V.$$

For Graph Neural Networks we consider graphs $G = (V, E)$, together with node features, where each node $v \in V$ is associated with a feature vector $x_v \in \mathbb{R}^d$ for $d \in \mathbb{N}$. The x_v, by convention, are row vectors. In matrix notation with $V = \{v_1, ..., v_n\}$, we have the *feature matrix*

$$\mathbb{R}^{n \times d} \ni X := \begin{pmatrix} x_{v_1} \\ \vdots \\ x_{v_n} \end{pmatrix}.$$

Graphs are very general and highly complex data structures, which can be seen as a generalization of text (1D sequences) or images (2D grids), thus one needs specialized neural models, to deal with them. On the upside, however, because of their generality, one can express a lot of data, and especially relationships between different parts of said data, as graphs. Typical examples of data that lends itself to representation via a graph are social networks or citation networks, as well as complex molecules. The idea for graph neural networks stems from the popular and highly successful CNN models for images, which gather local data, to form highly expressive representations [Zho+21]. The power of graph neural networks stems from the unity of the expressive power of graph data with the local convolutions of CNNs.
When studying Graph Neural Networks, one distinguishes two main tasks:

1. Node Classification:
   In node classification, one wants to predict a class for every node of an oftentimes large graph. A typical example would be the classification of people in a social network based on their friends for targeted advertisement, or the classification of scientific publications, with the links in a graph being citations. This is also what we will do, as Cora, Pubmed and CiteSeer are so-called citation networks. Node classification is done by stacking some GNN layers, as they are described below, with possibly a neural network on top of the last representation, followed by a softmax.

2. Graph Classification:
   In graph classification, one wants to predict a single class for the entire graph. Since the GNN layers output a representation for each node of the graph, one needs an extra layer to combine all those representations into a single representation for the whole graph. For this, so-called pooling layers are used (see Section 2.1.5). After that, one can proceed as above, with a neural network and softmax output. Examples for graph classification are the classification of text in natural language processing, as a text can be transformed into a graph, or to predict properties of molecules. This also is the task for the MUTAG dataset.

Now we want to review a small selection of Graph Neural Network layers. For a more complete list, see [Zho+21]. In the following, let $G = (V, E)$ be a graph with adjacency matrix A and feature matrix X. Most Convolutional Graph Neural Networks work similarly. They first locally aggregate the features of the graph in some way, using the graph structure, and then apply a weight matrix and some non-linear function. In matrix notation, we can write

$$H^{(k)} = \sigma\left(AGG\left[A, H^{(k-1)}\right] W^{(k)}\right), \tag{1}$$

where σ is some activation function, $H^{(k)}$ is the k-th *feature representation*, with $H^{(0)} = X$, $W^{(k)}$ is the k-th *weight matrix* and AGG is the aggregation mechanism [Rat+21, Equation (1)].
For a node $v \in V$ let $h_v^{(k-1)}$ be the row vector of $H^{(k-1)}$, that is associated with v. Also, let $\hat{A} := A + I$ and $\hat{D} := D + I$, where I is the identity matrix of degree $|V|$, A is the adjacency matrix of G, and D is the degree matrix of G. Now, we will introduce some of the most important aggregation mechanisms.

2.1.1 Graph Convolutional Network

First, we consider *Graph Convolutional Networks (GCN)* [KW17], since they are one of the most common types of GNN layers.
The aggregation step is

$$h_v^{(k-1)'} = \sum_{u \in \mathcal{N}(v) \cup \{v\}} \frac{1}{\sqrt{(\deg u + 1)(\deg v + 1)}} h_u^{(k-1)}.$$

The aggregated representation is a weighted sum of the representations of neighboring nodes, weighted by node degrees. Then, the matrix formulation of the aggregation step is

$$H^{(k-1)'} = \hat{D}^{-\frac{1}{2}} \hat{A} \hat{D}^{-\frac{1}{2}} H^{(k-1)}$$

and the whole update step (Equation (1)) becomes

$$H^{(k)} = \sigma\left(\hat{D}^{-\frac{1}{2}} \hat{A} \hat{D}^{-\frac{1}{2}} H^{(k-1)} W^{(k)}\right).$$

2.1.2 Graph Attention Network

In contrast to the GCNs, the Graph Attention Network [Vel+18] uses attention weights in the aggregation step to control the influence of neighboring nodes on the overall sum. For some node $v \in V$ with neighborhood $\mathcal{N}(v)$ the aggregation is

$$h_v^{(k-1)'} = \sum_{u \in \mathcal{N}(v) \cup \{v\}} \alpha_{v,u}^{(k-1)} h_u^{(k-1)},$$

with

$$\alpha_{v,u}^{(k-1)} = \frac{\exp \varphi \left(a_{(k-1)}^T \left[h_v^{(k-1)} || h_u^{(k-1)} \right] \right)}{\sum_{u \in \mathcal{N}(v) \cup \{v\}} \exp \varphi \left(a_{(k-1)}^T \left[h_v^{(k-1)} || h_u^{(k-1)} \right] \right)}.$$

Here $a_{(k-1)}$ is a vector of weights, φ is the leaky ReLU function and $||$ is the concatenation operator:

$$\mathbb{R}^n \times \mathbb{R}^m \ni ((a_1, ..., a_n), (b_1, ..., b_m)) \mapsto (a_1, ..., a_n)||(b_1, ..., b_m) = (a_1, ..., a_n, b_1, ..., b_m) \in R^{n+m}.$$

2.1.3 Simple Graph Convolution

The simple graph convolution (SGN) [Wu+19] is very similar to GCN. The only difference is, that the non-linear function σ is omitted, instead, the only non-linear part is the aggregation itself. Therefore, the update step is

$$H^{(k)} = \hat{D}^{-\frac{1}{2}} \hat{A} \hat{D}^{-\frac{1}{2}} H^{(k-1)} W^{(k)}.$$

This makes it very straightforward to implement deep networks made up of SGNs, since the k level update step is

$$H^{(k)} = \left(\hat{D}^{-\frac{1}{2}} \hat{A} \hat{D}^{-\frac{1}{2}} \right)^k X \Theta,$$

with $\Theta = W^{(k)} W^{(k-1)} \cdots W^{(1)}$.

2.1.4 GraphConv

The aggregation method of GraphConv [Mor+20] is again a successor to GCNs, which distinguishes the node from its neighborhood by using different weights for the neighborhood and the node in question, where in GCN the complete aggregation is performed before multiplying by the weights.
The update step of GraphConv is

$$h_v^{(k)} = \sigma \left(h_v^{(k-1)} W_1^{(k)} + \sum_{u \in \mathcal{N}(v)} h_u^{(k-1)} W_2^{(k)} \right)$$

for $v \in V$, or

$$H^{(k)} = \sigma \left(H^{(k-1)} W_1^{(k)} + A H^{(k-1)} W_2^{(k)} \right)$$

in matrix notation.

2.1.5 Pooling

When wanting to do graph classification, one needs a way of combining representations of every node of a graph to a single representation of the graph as a whole. For this purpose, pooling layers are used.
The most common of which are the following:

- Mean pooling or average pooling (GAP; global average pooling):
  The representation of the graph is the average of the node representations:
  $$h_{\text{pool}} = \frac{1}{|V|} \sum_{v \in V} h_v^{(k)}.$$
- Max pooling:
  The graph representation is the component-wise maximum of the node representations:
  $$h_{\text{pool}} = \max_{v \in V} h_v^{(k)}.$$
- Attention pooling (GAT; global attention pooling) [Li+17]:
  The graph representation is a weighted sum of the node representations:
  $$h_{\text{pool}} = \sum_{v \in V} \alpha_v h_v^{(k)}$$
  with
  $$(\alpha_v)_{v \in V} = \text{softmax}\left[\left(NN(h_v^{(k)})\right)_{v \in V}\right],$$
  where NN is a neural network.

2.2 Gradient-based Attribution

Attribution characterizes a class of techniques, that try to explain the decisions of a neural model by weighting the inputs of the model based on how important they are for the model's decision. In the following, we consider different attribution techniques based on the list in [San+20]. These have in common, that they produce node-level attribution weights. This means that, at least in the versions of the techniques we are using, every node of the graph gets weighted by how important it is to the decision. For some of these techniques, one could also look at how important every single feature of every node is and also which edges and edge features are important, but for comparability and clarity, we only consider attention weights for the nodes as a whole. The first few of the following techniques only rely on derivatives of the network's output with respect to the input and are therefore rather straightforward to implement for various types of neural networks, while CAM (Section 2.2.5) and GradCAM (Section 2.2.6) are not model agnostic. Specifically, they need certain types of pooling layers. We will use them for the graph classification task.
In the following, let $x_v = (x_{v,1}, ..., x_{v,d}) \in \mathbb{R}^d$ be the feature vector of a node $v \in V$ and let y be the networks output of the predicted class.

2.2.1 Gradients

The idea behind this technique is to look at the gradient of the output with respect to the inputs. The more the output changes, when changing the input, the more important this specific input is to the prediction, while inputs with smaller gradients do barely change the prediction, and are therefore less important to the network.
Now, the gradients by themselves are attributions of the features. To receive attribution weights for the node, we use the euclidean norm of the gradients with respect to the features of the node. The attribution weight of the node $v \in V$ is
$$w_v = \left\| \left(\frac{\partial y}{\partial x_{v,1}}, ..., \frac{\partial y}{\partial x_{v,d}} \right) \right\|_2.$$

2.2.2 GradInput

GradInput is the element-wise product of the inputs with the gradients from Section 2.2.1. It was first proposed in [Shr+17] and corresponds to the first-order Taylor approximation when the bias term is 0. Like before, we can use the norm to reduce this to the node-level attribution:

$$w_v = \left\| \left(x_{v,1} \frac{\partial y}{\partial x_{v,1}}, ..., x_{v,d} \frac{\partial y}{\partial x_{v,d}} \right) \right\|_2 .$$

2.2.3 SmoothGrad

SmoothGrad, proposed in [Smi+17], is a method, that aims at reducing noise and artifacts in GradInput by computing the GradInput value for multiple, slightly perturbed inputs. For GNNs, we perturb the input values, by adding noise to the node features, while leaving the topology of the graph constant. The noise is made up of independent samples from a normal distribution (see Definition 3.1.7) with $\mu = 0$ and σ in the range of $0.1 \cdot \max(input)$ to $0.2 \cdot \max(input)$. The final weights are the average weights from n iterations. Typically $n = 100$ is used. We first compute attribution weights for each feature, which then get reduced to node-level attributions by the norm:

$$w_v = \frac{1}{n} \left\| \sum_{j=1}^{n} \left(x_{v,1} \frac{\partial y}{\partial x_{v,1}} (G + noise_j), ..., x_{v,d} \frac{\partial y}{\partial x_{v,d}} (G + noise_j) \right) \right\|_2 .$$

Here, $G + noise_j$ is the graph G, but $noise_j$ is added to the node features with $noise_j$ being a matrix of independent samples of the normal distribution described above, with the same shape as X.

2.2.4 IntegratedGradients

IntegratedGradients [STY17] not only considers the local changes, like GradInput does. Instead, it integrates all changes from some baseline G' to the graph G we want to evaluate. For images, this baseline would simply be a black image of the corresponding dimensions. For graphs, the question of a baseline is not that simple, as there is no such thing as the one baseline graph. The most simple baseline, which would carry no information about the original graph, would be just one node with a feature vector of zero, however then, a smooth transition from the baseline G' to the original graph G would be impossible, and we would end up in a situation where we would need an intermediate graph with maybe four nodes, or half an edge. Even before trying to interpret, what half an edge could be, there is the question of which four nodes from the original graph one should take. That is why, in the domain of GNNs, the baseline G' for the graph G is the same graph, but with node features of zero. The weight of some feature $x_{v,i}$ of the node $v \in V$ then is

$$w_{v,i} = x_{v,i} \int_0^1 \frac{\partial y}{\partial x_{v,i}} (G' + \alpha(G - G')) d\alpha,$$

with $G' + \alpha(G - G')$ being topologically the same graph as G, but with feature matrix

$$X_{G'+\alpha(G-G')} = \underbrace{X_{G'}}_{=0} + \alpha(X_G - \underbrace{X_{G'}}_{=0}) = \alpha X_G,$$

and $y(G' + \alpha(G - G'))$ being the networks output of the predicted class of G. The weight of the node $v \in V$ is again

$$w_v = \|w_{v,1}, ..., w_{v,d}\|_2 .$$

2.2.5 CAM

Class Action Mapping (CAM) [Zho+15] is a way of evaluating attribution in the graph classification task when using global average pooling. It works by weighting the nodes based on how closely their representations are aligned with the average.
Let $h_v^{(k)}$ be the node representation of the node $v \in V$ after the last GNN layer and

$$h_{\text{pool}} = \frac{1}{|V|} \sum_{v \in V} h_v^{(k)}$$

the output of the average pooling layer. Then the attribution weight of a node $v \in V$ is

$$w_v = \langle h_v^{(k)}, h_{\text{pool}} \rangle,$$

where $\langle \cdot, \cdot \rangle$ is the standard scalar product.

2.2.6 GradCAM

GradCAM [Sel+19] is an extension of CAM to other pooling layers. Instead of the outputs of the last GNN layer, one uses the gradients with respect to those outputs.
Let again $h_v^{(k)}$ be the node representation of the node $v \in V$ after the last GNN layer and h_{pool} be the output of the pooling layer. The GradCAM attribution weight of $v \in V$ is

$$w_v = \langle \frac{\partial y}{\partial h_v^{(k)}}, h_{\text{pool}} \rangle.$$

2.3 KEdge

In contrast to the gradient-based methods, that produce post-hoc explanations of an already-trained network, KEdge is an interpretable by design GNN layer. The idea is to generate a binary mask Z for the adjacency matrix A, to mask out some edges of the input graph. This produces the new, sparsified adjacency matrix

$$A_{sprs} = A \odot Z,$$

where $\odot$ is the pointwise multiplication operator. Using that, we can simply parse the graph together with this new adjacency matrix through the GNN. Now one can use the mask Z to interpret the GNN. Edges with a mask of approximately 1 are more important to the network's decision than edges with lower mask values, while edges with mask 0 do not affect the model's results and are therefore not important to the model's decision [Rat+21]. Equivalently, nodes which have larger mask values in most incident edges are more important to the model than nodes that tend to have lower masks of incident edges (see Section 3.2).
KEdge works by sampling the matrix Z from some probability distribution on $[0, 1]$ during training. We choose a parametrized class of distributions on $[0, 1]$ (see Section 3.1), such that every distribution of the class is continuous inside $(0, 1)$ and has positive point probabilities at zero and one. These distributions are used to approximate a Bernoulli distribution, so for every $p \in [0, 1]$, there should be a choice of parameters, such that the resulting distribution approximates the Bernoulli distribution with probability p reasonably well.
Let $||$ again be the concatenation operator and for some node $u \in V$ let $x_u \in \mathbb{R}^d$ be the feature vector of that node. To obtain some parameter α of the distribution for a given edge $e = (u, v) \in E$ we use an attention-like mechanism[1]:

$$\alpha_{u,v} = \exp\left[-\sigma\left(\theta_\alpha^T \left[W_\alpha x_u || W_\alpha x_v\right]\right)\right], \tag{2}$$

[1]We take the algorithm from the reference implementation of KEdge and not from [Rat+21] directly. See Remark 2.3.1.

with $W_\alpha \in \mathbb{R}^{d' \times d}$, $\theta_\alpha \in R^{2d'}$.
We only use distributions, that have two parameters, and therefore do this calculation independently for both parameters, with each one getting its own weights W and θ.
When using a sampling mechanism that is differentiable with respect to the distribution's parameters, we can use the backpropagation algorithm to learn the network's parameters, as well as the additional parameters θ_α and W_α, at the same time.
We then train all parameters at the same time using the new loss function

$$\mathcal{L}_{KEdge}(X, A, \Theta_{orig}, \Theta_{att}) = \mathcal{L}_{orig}(X, A_{sprs}, \Theta_{orig}) + \lambda \mathbb{E}[\|Z\|_0 \,|X, A, \Theta_{att}], \tag{3}$$

where X is the feature matrix, A is the adjacency matrix, Θ_{orig} is the set of parameters of the original model, and Θ_{att} is the set of parameters of the attention mechanism in Equation (2). $\|Z\|_0$ is the so-called L^0-norm of Z; that is the number of entries that are *not* equal to zero. The expectation term in Equation (3) then just becomes

$$\mathbb{E}[\|Z\|_0 \,|X, A, \Theta_{att}] = \sum_{e=(u,v)\in E} 1 - \mathbb{P}[Z_{u,v} = 0 | X, A, \Theta_{att}]$$

where $Z_{u,v}$ is distributed by our distribution of choice with parameters according to Equation (2). The parameter $\lambda \in \mathbb{R}$ in Equation (3), called the *maskedness* parameter, is a hyperparameter that controls how many edges are pruned from the original graph.
At test time we stop sampling the matrix Z and instead choose its entries $Z_{u,v}$ deterministically as zero, $\mathbb{E}[Z_{u,v}]$, or one depending on which of $\mathbb{P}[Z_{u,v} = 0]$, $\mathbb{P}[0 < Z_{u,v} < 1]$, or $\mathbb{P}[Z_{u,v} = 1]$ is the largest. Alternatively, if a hard mask is required, we can also just choose zero or one, depending on which probability is higher.

Remark 2.3.1:
We revert to the reference implementation from KEdge, and *not* the algorithm from the paper [Rat+21] itself, since that would involve the distribution's parameters to be calculated via the softmax function

$$\alpha_{u,v} = \frac{\exp\left[\sigma\left(\theta_\alpha^T \left[W_\alpha x_u || W_\alpha x_v\right]\right)\right]}{\sum_{w \in \mathcal{N}(u)} \exp\left[\sigma\left(\theta_\alpha^T \left[W_\alpha x_u || W_\alpha x_w\right]\right)\right]}.$$

This would not only limit the expressive range of the parameters to the interval $(0, 1)$, but it would also put all the edges incident to the node u in some kind of competition over the parameters' values, as the parameter for one edge being large would automatically mean that the parameters corresponding to the other edges will be smaller. Depending on the influence of a given parameter on the distribution, this leads to a huge bias for or against the importance of all edges, which only allows for a small number of exceptions. This would also make it almost impossible to control the number of removed edges via the maskedness parameter λ.
Going one step further, one could also calculate the parameters via any neural network NN with appropriately sized layers;

$$\alpha_{u,v} = NN\left[W_\alpha x_u || W_\alpha x_v\right].$$

However, our experiments show, that this would not lead to better results, since the expressive power of Equation (2) is enough to lead to appropriate values for the distribution's parameters, given a suitable value for λ, since these will be condensed to one of the three options zero, one, or $\mathbb{E}[Z_{u,v}]$ at test time anyways.

Remark 2.3.2:
The attributions gained from KEdge depend only on the input graph. Therefore, they are local attributions in the graph classification task, since the attribution is unique for each graph. In

the case of node classification, however, the attributions are global, since they do not depend on which node we want to classify, but they are rendered for the whole dataset (a single, typically large graph) before anything else happens.
This is in contrast to the gradient-based methods from Section 2.2, which always produce local explanations. We therefore also expect KEdge to provide poorer explanations when compared to the local explanations of the gradient-based methods in the node classification task. The advantage of global explanations is, however, that they are broader.

3 Methods

In this section, we introduce all the tools, we need for our experiments. First, we define a bunch of probability distributions, which we can then use and evaluate together with KEdge. It should be noted, that the parameter generation in KEdge only produces positive parameters. This is not a problem for most distributions and even desirable for some, like the Beta (Definition 3.1.12) and Kuma (Definition 3.1.13) distributions. For other distributions, however, this is the reason, that we cannot mask out any edges, using the plain algorithm. The Gumbel distribution, for example, is skewed to the right, so that even for low $\mu > 0$ it is very unlikely to sample zero in the HardGumbel distribution. In that case, we use an additional transformation to make the whole range of possible parameter values attainable. Our tests showed, that the function

$$f(x) = \log(x + \varepsilon),$$

for some small $\varepsilon > 0$, is suitable for this and does not introduce any additional bias to the parameters' values, as a simple, linear transformation would.
After defining all the different probability distributions we need, we also establish three ways of casting the edge weights, KEdge produces, to node weights, which are easier to make sense of (Section 3.2). These we can compare to the attribution weights of the gradient-based methods in our experiments.
As a third step in this chapter (Section 3.3), we discuss some metrics for the evaluation of attribution weights and we will also define our new metric; the integrated fidelity score.

3.1 Probability Distributions

For KEdge we want to construct approximations of the Bernoulli distribution. As a basis for the construction, we will take some distributions, which are absolutely continuous with respect to the Lebesgue measure, and modify them a bit, to receive distributions on $[0, 1]$ with positive point probabilities at zero and one.
In the following, we describe distributions as measures on the Borel sigma-algebra $\mathcal{B}(\mathbb{R})$ using the following notation.

Notation 3.1.1:
For some measure μ on $\mathcal{B}(\mathbb{R})$ and some measurable density f, we define a new measure λ on $\mathcal{B}(\mathbb{R})$ by

$$\lambda(A) := \int_A f(x) d\mu(x)$$

for all $A \in \mathcal{B}(\mathbb{R})$.
Using this, we can define the notation

$$f d\mu := \lambda.$$

Remark 3.1.2:
Let $(E, \mathcal{E})$ be a measurable space. From the Radon-Nikodym theorem, we know that such a density f exists for the measures λ and μ, if and only if λ is absolutely continuous with respect to μ, that is $\mu(A) = 0$ implies $\lambda(A) = 0$ for all $A \in \mathcal{E}$ [Rud87, Thm. 6.10].

Now, we quickly define some notions regarding random variables and distributions.

Definition 3.1.3 (random variable, CDF, PDF)**:**
Let $(E, \mathcal{E})$ be a measurable space and $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space.
A *random variable* X is a measurable map $X : \Omega \to E$.

Let μ be the measure on E defined by

$$\mu(A) := \mathbb{P}[X \in A]$$

for all $A \in \mathcal{E}$. Then we call μ the *distribution* of X and write $X \sim \mu$.
If $E = \mathbb{R}$ and $\mathcal{B}(\mathbb{R}) \subset \mathcal{E}$, we call

$$F(x) := \mathbb{P}[X \leq x] = \mu((-\infty, x])$$

the *cumulative density function (CDF)* of X or μ respectively.
If furthermore μ is absolutely continuous with respect to the Lebesgue measure λ, we call the Radon-Nikodym density

$$f := \frac{\partial \mu}{\partial \lambda},$$

the *probability distribution function (PDF)* of X or μ respectively.
In this case, we also have

$$f = \frac{\partial F}{\partial x} \qquad \lambda\text{-almost surely}$$

For KEdge, we require the distributions and sample functions to be differentiable with respect to the parameters. This is implemented through the so-called reparametrization trick:
When we consider a probability distribution D_Θ with parameter $\Theta \in \mathbb{R}^d$, we reparametrize by considering another distribution Δ and function f, such that

$$f(\Theta, X) \sim D_\Theta \text{ for } X \sim \Delta$$

and f is differentiable with respect to the parameter Θ Δ-almost surely.
Hereinafter, let $\delta_x := \mathbb{1}_{\{x\}}$ be the point measure in $x \in \mathbb{R}$.

3.1.1 Distributions on $\mathbb{R}$

To transform distributions from $\mathbb{R}$ to $[0, 1]$, we simply pass the resulting random variables through the following cutoff function:

$$\mathbb{R} \ni x \mapsto \min(1, \max(x, 0)) \in [0, 1].$$

Lemma 3.1.4 (hard transformation)**:**
Let $(E, \mathcal{E})$ be a measurable space, $(\Omega, \mathcal{F}, \mathbb{P})$ a probability space and $X : \Omega \to \mathbb{R}$ a random variable with a distribution, that is absolutely continuous with respect to the Lebesgue measure, CDF F_X, and PDF f_X. Then the random variable

$$Y := \min(1, \max(X, 0))$$

possesses the distribution

$$\nu_Y = F_X(0)d\delta_0 + \mathbb{1}_{(0,1)} f_X dx + (1 - F_X(1))d\delta_1$$

and the CDF

$$F_Y(x) = \begin{cases} 0, & x < 0 \\ F_X(x), & 0 \leq x < 1 \\ 1, & x \geq 1 \end{cases}.$$

Proof. Let $A \in \mathcal{B}(\mathbb{R})$. Then we have

$$\mathbb{P}[Y \in A \cap (0,1)] = \mathbb{P}[X \in A \cap (0,1)] = \int\limits_{A \cap (0,1)} f_X(x)dx.$$

We also have

$$\mathbb{P}[Y \in (-\infty, 0) \cup (1, \infty)] = \mathbb{P}(\emptyset) = 0$$

and

$$\mathbb{P}[Y = 0] = \mathbb{P}[X \leq 0] = F_X(0),$$

as well as

$$\mathbb{P}[Y = 1] = \mathbb{P}[X \geq 1] = \underbrace{\mathbb{P}[X = 1]}_{=0} + \underbrace{\mathbb{P}[X > 1]}_{1-F_X(1)} = 1 - F_X(1).$$

All in all

$$\begin{aligned}\mathbb{P}[Y \in A] =& \underbrace{\mathbb{P}[Y \in A \cap (-\infty, 0)]}_{=0} + \underbrace{\mathbb{P}[Y \in A \cap \{0\}]}_{=F_X(0)\delta_0(A)} + \underbrace{\mathbb{P}[Y \in A \cap (0,1)]}_{=\mathbb{P}[X \in A \cap (0,1)]} \\ &+ \underbrace{\mathbb{P}[Y \in A \cap \{1\}]}_{=(1-F_X(1))\delta_1(A)} + \underbrace{\mathbb{P}[Y \in A \cap (1, \infty)]}_{=0} \\ =& \int\limits_A F_X(0)d\delta_0 + \mathbb{1}_{(0,1)} f_X dx + (1 - F_X(1))d\delta_1.\end{aligned}$$

□

Lemma 3.1.4 is visualized in Figure 1.

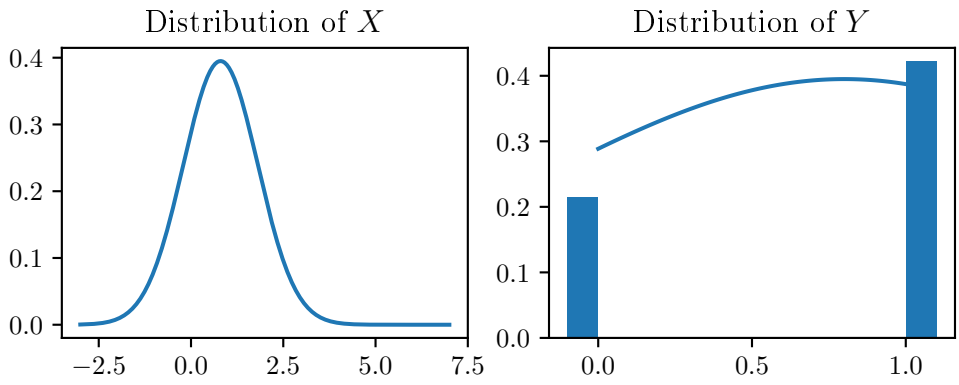


Figure 1: Hard transformation: PDF of X and resulting PDF of $Y = \min(1, \max(0, X))$ in $(0, 1)$ with point probabilities at zero and one.

It is used to transform the following distributions:

Definition 3.1.5 (Cauchy, HardCauchy)**:**
We define the *Cauchy* distribution with parameters $t \in \mathbb{R}$ and $s > 0$ to be the measure

$$\nu_{cauchy} = \frac{1}{\pi} \frac{s}{s^2 + (x - t)^2} dx.$$

Using the transformation from Lemma 3.1.4, we receive the *HardCauchy* distribution.

Definition 3.1.6 (Laplace, HardLaplace)**:**
We define the *Laplace* distribution with parameters $\mu \in \mathbb{R}$ and $b > 0$ to be the measure

$$\nu_{laplace} = \frac{1}{2b} \exp\left(-\frac{|x-\mu|}{b}\right) dx.$$

The transformation in Lemma 3.1.4 then produces the *HardLaplace* distribution.

Definition 3.1.7 (Normal, HardNormal)**:**
We define the *Normal* distribution with parameters $\mu \in \mathbb{R}$ and $\sigma > 0$ to be the measure

$$\nu_{normal} = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) dx.$$

The transformation in Lemma 3.1.4 then yields the *HardNormal* distribution.

Definition 3.1.8 (Gumbel, HardGumbel)**:**
We define the *Gumbel* distribution on $\mathbb{R}$ with parameters $\beta > 0$ and $\mu \in \mathbb{R}$ to be the measure

$$\nu_{gumbel} = \frac{1}{\beta} e^{-\frac{x-\mu}{\beta}} e^{-e^{-\frac{x-\mu}{\beta}}} dx.$$

Using Lemma 3.1.4 gives the HardGumbel distribution.

3.1.2 Distributions on $\mathbb{R}^+$

With distributions on $\mathbb{R}^+$, we can't just use Lemma 3.1.4, since the goal is to receive a random variable with a positive point probability at zero. We can, however, just translate the variable by a little, to receive a positive probability for $X < 0$, using the next lemma:

Lemma 3.1.9:
Let X be a random variable on $\mathbb{R}$ with CDF F_X and $\lambda \in \mathbb{R}$. Then the random variable $X + \lambda$ has CDF

$$F_{X+\lambda}(x) = F_X(x - \lambda).$$

In case the distribution of X is absolutely continuous with respect to the Lebesgue measure, and if X has PDF f_X, then $X + \lambda$ has PDF $x \mapsto f_X(x - \lambda)$.

Proof. This is just the simple fact, that

$$F_{X+\lambda}(x) = \mathbb{P}[X + \lambda \leq x] = \mathbb{P}[X \leq x - \lambda] = F_X(x - \lambda)$$

and

$$f_{X+\lambda}(x) = \frac{\partial F_{X+\lambda}}{\partial x}(x) = \frac{\partial F_X}{\partial x}(x - \lambda) = f_X(x - \lambda) \qquad a.s.$$

□

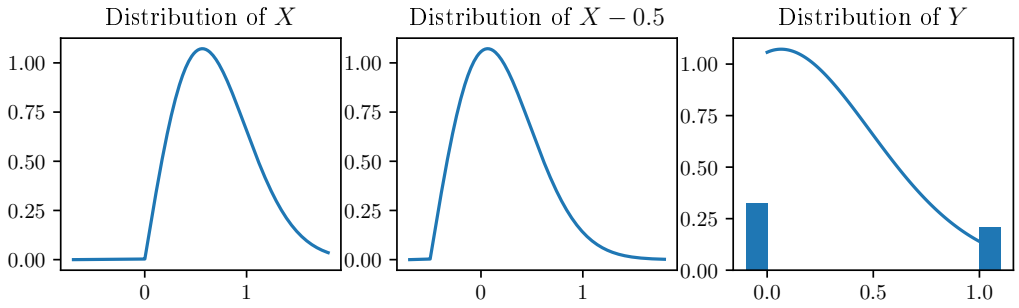


Figure 2: Translation: PDF of X, PDF of $X - 0.5$, and resulting PDF of $Y = \min(1, \max(0, X - 0.5))$ in $(0, 1)$ with point probabilities at zero and one.

Figure 2 visualizes the transformation of a positive random variable X, that is absolutely continuous with respect to the Lebesgue measure, to the corresponding hard random variable using Lemma 3.1.4 and Lemma 3.1.9. We use the same method to transform the following distribution:

Definition 3.1.10 (Weibull, HardWeibull)**:**
We define the *Weibull* distribution with parameters $\lambda, k > 0$ to be the measure

$$\nu_{weibull} = \mathbb{1}_{[x>0]} \frac{k}{\lambda} \left(\frac{x}{\lambda}\right)^{k-1} \exp\left(-\left(\frac{x}{\lambda}^k\right)\right) dx.$$

Translating the Weibull distribution by $\kappa < 0$, using Lemma 3.1.9, and then using Lemma 3.1.4 yields the *HardWeibull* distribution on $[0, 1]$.

3.1.3 Distributions on $(0, 1)$

For distributions on $(0, 1)$ we use a similar trick as in the $\mathbb{R}^+$ case. The difference is, that now we will stretch the support, instead of only translating it, using the function

$$x \mapsto a + (b - a)x,$$

which maps $(0, 1)$ to (a, b). We will set $a < 0$ and $b > 1$, to then apply the hard transformation of Lemma 3.1.4.

Lemma 3.1.11:
Let X be a random variable on $\mathbb{R}$ with CDF F_X. Then $Y := a + (b - a)X$ has CDF

$$F_Y(x) = F_X\left(\frac{x - a}{b - a}\right).$$

If additionally the distribution of X is absolutely continuous with respect to the Lebesgue measure and X has PDF f_X, then Y has PDF

$$f_Y = \frac{1}{b - a} f_X\left(\frac{x - a}{b - a}\right).$$

Proof. We have

$$F_Y(x) = \mathbb{P}[Y \leq x] = \mathbb{P}[a + (b - a)X \leq x] = \mathbb{P}\left[X \leq \frac{x - a}{b - a}\right] = F_X\left(\frac{x - a}{b - 1}\right)$$

and

$$f_Y(x) = \frac{\partial}{\partial x} F_Y(x) = \frac{\partial}{\partial x} F_X\left(\frac{x - a}{b - a}\right) = \frac{1}{b - a} f_X\left(\frac{x - a}{b - a}\right).$$

□

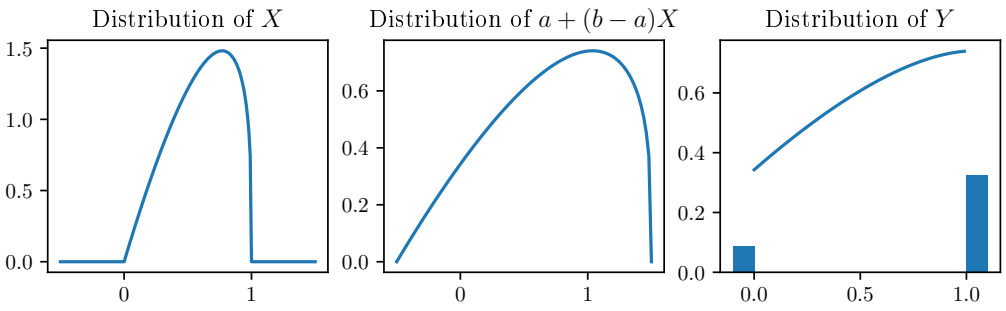


Figure 3: Linear Transformation: PDF of X, PDF of $a + (b - a)X$, and resulting PDF of $Y = \min(1, \max(0, a + (b - a)X))$ in $(0, 1)$ with point probabilities at zero and one for $a = -\frac{1}{2}$ and $b = \frac{3}{2}$.

Figure 3 visualizes the transformation of a random variable $X \in (0,1)$, which is absolutely continuous with respect to the Lebesgue measure, to the corresponding hard random variable. This method uses Lemma 3.1.11 and Lemma 3.1.4. We use this to transform the following distributions:

Definition 3.1.12 (Beta, HardBeta)**:**
We define the *Beta* distribution with parameters $\alpha, \beta > 0$ to be the measure

$$\nu_{beta} = \mathbb{1}_{0<x<1} \frac{1}{B(\alpha,\beta)} x^{\alpha-1}(1-x)^{\beta-1} dx,$$

where B is the beta function

$$B(\alpha,\beta) = \int_0^1 x^{\alpha-1}(1-x)^{\beta-1} dx.$$

Using the transformation from Lemma 3.1.11 for some $a < 0$ and $b > 0$, followed by the transformation from Lemma 3.1.4 yields the *HardBeta* distribution.

Definition 3.1.13 (Kuma, HardKuma)**:**
We define the *Kumaraswamy (Kuma)* distribution with parameters $\alpha, \beta > 0$ to be the measure

$$\nu_{kuma} = \mathbb{1}_{0<x<1} abx^{a-1}(1-x^a)^{b-1} dx.$$

Using the transformation from Lemma 3.1.11 for some $a < 0$ and $b > 0$, followed by the transformation from Lemma 3.1.4 yields the *HardKuma* distribution.

3.2 Attribution from KEdge

While KEdge yields edge weights, we want to evaluate node weights, so we need a way to transform these edge weights to node weights. For an edge $e \in E$ let w_e be the edge weight. For a node $v \in V$ let $\mathrm{In}(v)$ be the set of incident edges on v (see Definition 2.1.1). Using this notation, we define different ways of transforming the edge weights to node weights.
First, we simply use the sum of the incident edge weights. The factor $\frac{1}{2}$ is there, simply to ensure that the sum of the edge weights equals the sum of the node weights:

$$w_v := \frac{1}{2} \sum_{e \in \mathrm{In}(v)} w_e. \tag{4}$$

The second way of transformation is using the mean instead of the sum. This can be useful so that we don't discriminate based on node degree. The corresponding weight then is

$$w_v := \frac{1}{\deg v} \sum_{e \in \mathrm{In}(v)} w_e. \tag{5}$$

Now it might be possible, that one important edge connects two important nodes, even though other edges of that node don't represent that. We therefore might try to reflect this in the node weights, using the maximum:

$$w_v := \max_{e \in \mathrm{In}(v)} w_e.$$

Remark 3.2.1:
Even though Sanchez-Lengeling et al. use the sum (Equation (4)) to redistribute edge weights to node weights, our experiments showed that using Equation (5) yields the best results most of the time.

3.3 Evaluating Attribution

Rationales as explanations for model prediction

To evaluate, how faithful the attribution techniques are to the model we want to explain, we compute the fidelity [FKA21] of the attribution weights. The fidelity measures how likely it is, that the prediction of a network changes, given that some node features stay the same, while others change. A high fidelity score means, that one selected a, preferably small, set of graph nodes, which play a large part in the decision making of the model.
We first define fidelity for binary attribution masks.

Definition 3.3.1 (fidelity [FKA21])**:**
Let G be an input graph with feature matrix X, NN be a neural model and A be a vector of binary attribution weights for each node of the graph. Let $c_0 := \operatorname{argmax} NN(G, X)$ be the predicted class of G, or of some node of G in case of node classification. The *fidelity* of the attribution A given the graph G and network NN is defined by

$$fid(A) := \mathbb{P}[NN(G, X_{rand}) = c_0 | (X_{rand})_{i,j} = X_{ij} \text{ for all } A_j = 1],$$

where the random feature matrix X_{rand} is made up of random features of X with uniform distribution.
The fidelity score is the probability, that the relevant network output stays the same, given the features of the nodes indicated in A stay constant.

A high fidelity score then means that the mask A selects the most important features of X.
Since the attribution masks of KEdge and those of the gradient-based attribution techniques are all soft masks, we cannot use the fidelity score as defined above. Therefore, we now extend the idea to soft masks:

Definition 3.3.2 ((integrated) fidelity)**:**
Let G be an input graph with feature matrix X, NN a neural model and A a vector of soft attribution weights for each node of the graph. Let $n := |V|$ be the number of nodes of G. The *(integrated) fidelity* of the attribution A given the graph G and network NN is

$$fid_{int}(A) := \int_0^1 fid(A_t) dt,$$

where A_t is the hard attribution mask, such that the $\lfloor tn \rfloor$ nodes with the highest attribution weights are chosen. When there are multiple nodes with the same weight, we could choose from, we take one at random.

Remark 3.3.3:
The (integrated) fidelity score does only depend on the order of attribution weights and not on the overall magnitude. Therefore, it is independent of monotone transformations, like normalization.

We now want to introduce two metrics that are similar to the integrated fidelity: The Sufficiency AOPC and Comprehensiveness AOPC from DeYoung et al. These are used to compare our results to the baseline models in the ERASER-benchmark paper [DeY+20]. These metrics were first developed for the text domain, but can also be extended to graphs. However, in our third experiment (Section 4.3), we use them on text, which we then convert to a graph.
We again first consider hard attribution weights (zero or one) to mark or highlight the important parts of the data. Here, sufficiency measures the goal of an explanation, that just the important, highlighted part is enough for the prediction of the class to stay constant, while comprehensiveness characterizes the fact that the model becomes less confident in the prediction of the data without the highlighted part.

Definition 3.3.4 (Comprehensiveness & Sufficiency)**:**
Let X be a random variable with values in a set $\mathcal{X}$ and let $f : \mathcal{X} \to [0,1]^m$ be a classifier for $m \in \mathbb{N}$ different classes. Let $R_X \in \mathcal{X}$ be a part of X that serves as an explanation and let $j_0 = \text{argmax}_{1 \leq j \leq m}[f_j(X)]$ be the predicted class. Then the *comprehensiveness* for the instance X is defined as

$$\text{comp}_X := f_{j_0}(X) - f_{j_0}(X \setminus R_X),$$

where $X \setminus R_X \in \mathcal{X}$ is the instance X, but with its part R_X removed. The *sufficiency* for the instance X is defined as

$$\text{suff}_X := f_{j_0}(X) - f_{j_0}(R_X).$$

Now we define the comprehensiveness of the model as the expected instance level comprehensiveness

$$\text{comp} := \mathbb{E}\left[\text{comp}_X\right] = \mathbb{E}\left[f_j(X) - f_j(X \setminus R_X)\right]$$

and the same with the models sufficiency

$$\text{suff} := \mathbb{E}\left[\text{suff}_X\right] = \mathbb{E}\left[f_j(X) - f_j(R_X)\right].$$

A high score in comprehensiveness means that the rationale R_X captures all or most of the important parts of the input and is what we want. In contrast, a low score in sufficiency means that the rationale R_X captures enough information so that f still predicts the class of X.
Now we do a similar trick of moving from soft attribution weights to hard masks.

Definition 3.3.5 (Comp. AOPC & Suff. AOPC)**:**
After generating our soft attribution mask, we generate five hard masks from it by selecting the highest $1\%, 5\%, 10\%$, 20%, and 50% of attribution scores. The *comprehensiveness area over the perturbation curve (AOPC)* and *sufficiency AOPC* are then defined to be the average of the comprehensiveness and sufficiency scores for each of those masks, respectively [DeY+20]:

$$\begin{aligned}\text{comp}\,AOPC := \frac{1}{5}\big(&\text{comp}(1\%\text{ mask}) + \text{comp}(5\%\text{ mask}) + \text{comp}(10\%\text{ mask}) \\ &+ \text{comp}(20\%\text{ mask}) + \text{comp}(50\%\text{ mask})\big)\end{aligned}$$

and

$$\begin{aligned}\text{suff}\,AOPC := \frac{1}{5}\big(&\text{suff}(1\%\text{ mask}) + \text{suff}(5\%\text{ mask}) + \text{suff}(10\%\text{ mask}) \\ &+ \text{suff}(20\%\text{ mask}) + \text{suff}(50\%\text{ mask})\big).\end{aligned}$$

Rationales as markers of what parts of the input are important

Another use of attribution weights is to check, what parts of the input are generally important and therefore should also be important to an effective model. Here we check whether soft attribution weights correspond to those parts of the input, deemed important by experts or humans in general. One such metric, that is also used by DeYoung et al. for the movie reviews dataset among others, is the area under the precision-recall-curve (AUPRC).
To explain AUPRC, we first define:

Definition 3.3.6 (Precision & Recall)**:**
Let X again be a random variable with values in $\mathcal{X}$ and let $\mathcal{X}$ be partitioned into two classes; positive $(+)$ and negative $(-)$. As a convention, one chooses $+$ to be the class where something interesting happens and $-$ to be the class with no effect (the null hypothesis). Let

$$f : \mathcal{X} \to \{+, -\}$$

be an estimator of the class of X and $c(X) \in \{+, -\}$ be the true class of X. We then define the *precision* of f to be

$$prec := \mathbb{P}[c(X) = +|f(X) = +]$$

the probability that X is of the positive class, given the estimation is positive, and the *recall* of f as

$$rec := \mathbb{P}[f(X) = +|c(X) = +]$$

the probability that the estimation is positive, given X is of the positive class.

Remark 3.3.7:
Precision and recall are closely related to the notions of errors of the first and second kind used in statistics. The error of the first kind is the mistaken rejection of the null hypothesis [Dek+05]. The probability of an error of the first kind is related to the precision:

$$\mathbb{P}[f(X) = + \text{ and } c(X) = -] = (1 - prec)\mathbb{P}[f(X) = +].$$

The error of the second kind is the mistake of *not* rejecting the null hypothesis, even though it is false [Dek+05]. This is even more closely related to the recall, since the normalizing constant of the recall does *not* depend on the estimator, but only on the dataset. The probability of an error of the second kind is:

$$\mathbb{P}[f(X) = - \text{ and } c(X) = +] = (1 - rec)\mathbb{P}[c(X) = +].$$

Now we can use this to define the precision-recall-curve:

Definition 3.3.8 (PRC, AUPRC)**:**
Let X be a random variable with values in $\mathcal{X}$ and let $\mathcal{X}$ be partitioned into two classes $+$ and $-$, where we choose $-$ to be the null hypothesis. Let $c(X)$ be the correct class of X and

$$f : \mathcal{X} \to \mathbb{R}$$

be an indication of the classes $+$ and $-$, such that a large $f(X)$ indicates that X belongs to $+$, while a small $f(X)$ indicates that X belongs to the negative class. From this we construct a family of estimators

$$f_c : \mathcal{X} \to \{+, -\},$$

where $c \in \mathbb{R}$ and

$$f_c(X) = \begin{cases} + & f(X) > c \\ - & f(X) \leq c \end{cases}.$$

Let $prec(c)$ be the precision of f_c and $rec(c)$ be the recall of f_c. The *precision-recall-curve (PRC)* is defined as

$$PRC := \{(rec(c), prec(c))|c \in \mathbb{R}\} \subset [0, 1]^2.$$

Now for $c \to \infty$, we get $f_c \equiv -$ and therefore $rec(c) = 0$, and for $c \to -\infty$, we get $f_c \equiv +$ and $rec(c) = 1$. Therefore, the precision-recall-curve always starts at a recall of zero and stops at a recall of one, while the precision falls from one to $\mathbb{P}[c(X) = +]$. Now the *area under the precision-recall-curve* is

$$AUPRC := \int_0^1 PRC(r)dr,$$

where $PRC(r) := prec(c_r)$ and c_r is chosen, such that $rec(c_r) = r$.

For our task of evaluating whether some generated annotations match human explanations, we work at a level of tokens. In our case (Section 4.3) these were words in a text, which we then converted to the nodes of a graph. For each token, we considered the human annotation, important (+) or not important (−), as the true class of a token. We then constructed the family $\{f_c\}_c \in \mathbb{R}$ of estimators from our annotated weights and computed the AUPRC as a metric of fit.

4 Experiments and Results

To answer the question, which technique delivers the best explanations, we conducted three experiments. At first, we evaluated the impact that including KEdge, with different levels of edge sparsity, in a GNN model has on the model's performance. In the second experiment, we compared explanations of different methods and judged which method provides good attributions and which one doesn't. In the last experiment, we visualized the attributions on a text dataset of movie reviews, from ERASER-benchmark [DeY+20], to judge the explanations by eye and evaluate different metrics for good attributions.
In our first two experiments, we used the citation networks CiteSeer, Cora and Pubmed [YCS16] for node classification and the protein dataset MUTAG [Sch+17] for graph classification. For a later experiment, we also used a modification of the movie reviews dataset [DeY+20]. See Table 2 for an overview of the datasets.
We also performed our experiments on the ogbg-molhiv dataset [Hu+21], but they weren't fruitful since that dataset has only 3.51% positive examples (but 96.49% negative ones). Because of that, the bias in our models became so large, that the prediction was negative, no matter the input. Then we of course saw no effect of the masked out edges on the performance and the fidelity of all models with all attribution techniques was 100% since the model did not switch prediction under any circumstance.

Dataset	Task	Nodes/Graphs	Features	Classes
CiteSeer	Node	3327	1433	6
Cora	Node	2708	3703	7
Pubmed	Node	19717	500	3
MUTAG	Graph	340	7	2
Movies	Graph	2000	300	2

Table 2: Statistics of the datasets. [YCS16; Sch+17; DeY+20]

For the node classification task, the model we used is made up of two GCN layers followed by a single, linear layer as a baseline and the same model with a KEdge layer, in the beginning, to calculate the edge weight first (Table 3). The intermediate size of KEdge is 32. In the graph classification task, we swapped the GCN layers for GraphConv layers and added a pooling layer. We used both global average and global attention pooling (Table 4). For the attention pooling, we used a simple two-layer network with an intermediate size of five in the calculation of the attention weights (see Section 2.1.5).

Operation	Feature size	Weights
Input	$features$	
KEdge		$2 \cdot (features \cdot 32 + 2 \cdot 32)$
GCN		$features \cdot 300 + 300$
ReLU	300	
GCN		$300 \cdot 200 + 200$
ReLU	200	
Linear		$200 \cdot classes + classes$
Softmax	$classes$	

Table 3: Model architecture for node classification.

For the movie reviews dataset, we used a slightly more complex model, which is explained in Table 7.

Operation	Feature size	Weights
Input	$features$	
KEdge		$2 \cdot (features \cdot 32 + 2 \cdot 32)$
GraphConv		$2 \cdot features \cdot 300 + 300$
ReLU	300	
GraphConv		$2 \cdot 300 \cdot 200 + 200$
ReLU	200	
pooling		GAP: 0 / GAT: $200 \cdot 5 + 5 + 5 \cdot 1 + 1$
Linear		$200 \cdot classes + classes$
Softmax	$classes$	

Table 4: Model architecture for graph classification.

We implemented the experiments using the PyTorch [Pas+19] and PyTorch Geometric [FL19] libraries. The implementation can be found in our git repository[2].

4.1 Performance of KEdge

With this first experiment, we want to answer the following:

Research Question 1:
How does the use of KEdge influence the performance of a GNN? What influence do different distributions and the maskedness parameter λ have on the performance?

Experiment

To answer this question, we first trained a model endowed with a KEdge layer. We trained a model for each distribution from Section 3.1 and each dataset while optimizing the maskedness parameter λ (Equation (3)) to attain different percentages of masked out edges. We then measured the test accuracy for each percentage of removed edges and compared the different distributions.

Results

One can clearly see a decreasing trend in the data points, which seem to be scattered around a line, as a first-order approximation[3]. We calculated the line of best fit to approximate the expected accuracy given a set percentage of removed edges. These approximations can now be compared to each other. The dashed line in each plot marks the accuracy of the baseline model without KEdge.

Node classification

For the node classification datasets, we could reliably reach every possible percentage of removed edges by optimizing the maskedness parameter λ in Equation (3).

[2] `https://git.l3s.uni-hannover.de/tfunke/tobias_interpretable_gnn.git`

[3] To check this for yourself, use *Evaluation/accuracy_ mask_ percent_ node_ classification.py* and *Evaluation/accuracy_ mask_ percent_ graph_ classification.py* from our git repository to generate that data and use the notebook *Evaluation/mask_ percent_ vs_ accuracy.ipynb* to plot it.

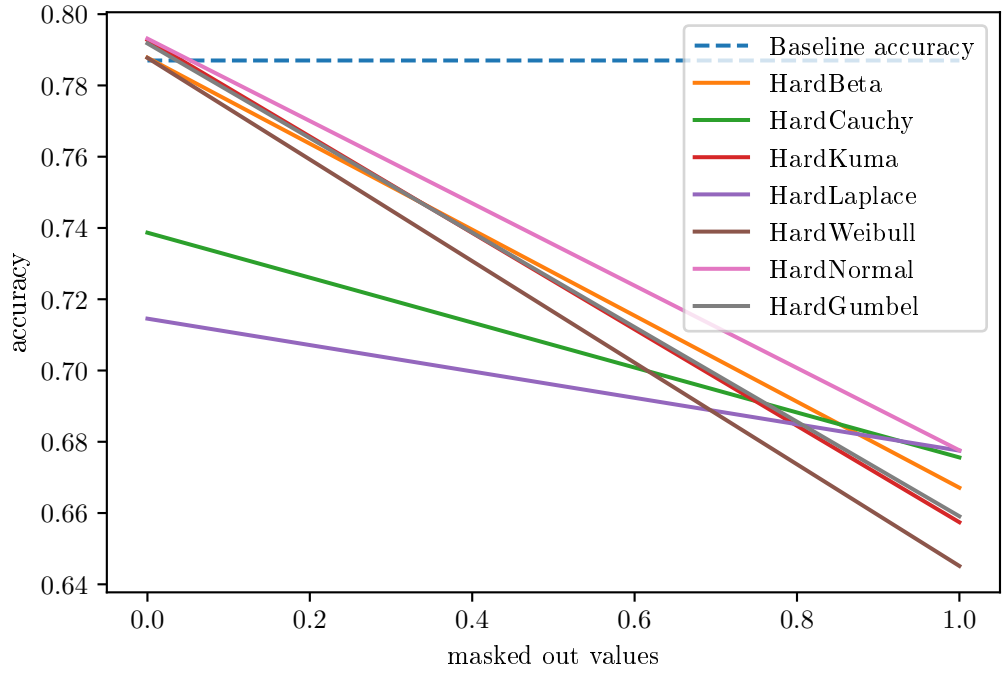


Figure 4: Linear approximation of the expected accuracy over mask out percentage for the Cora dataset

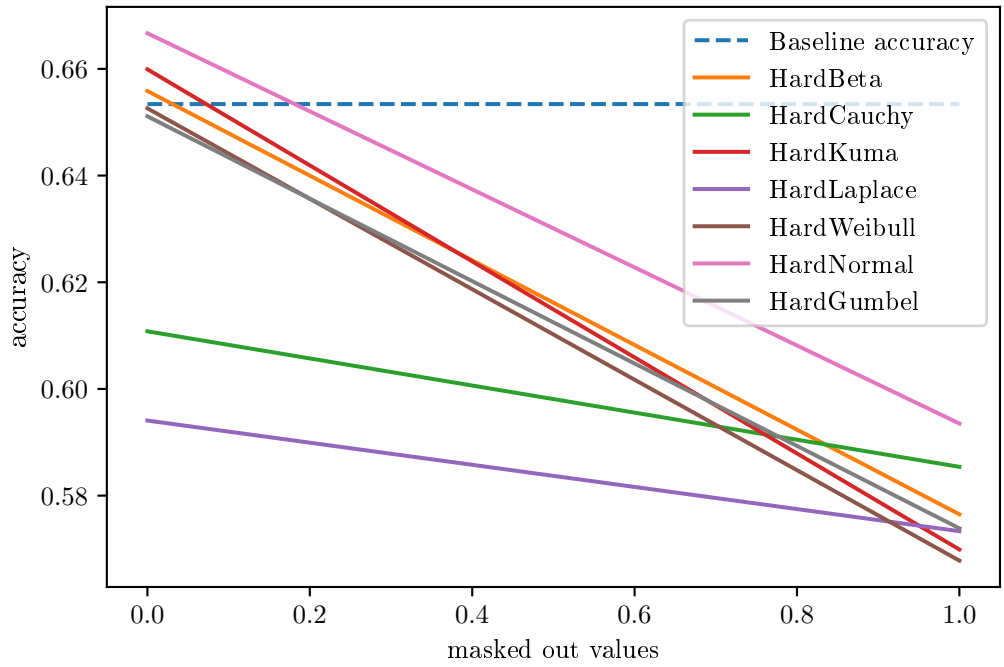


Figure 5: Linear approximation of the expected accuracy over mask out percentage for the CiteSeer dataset

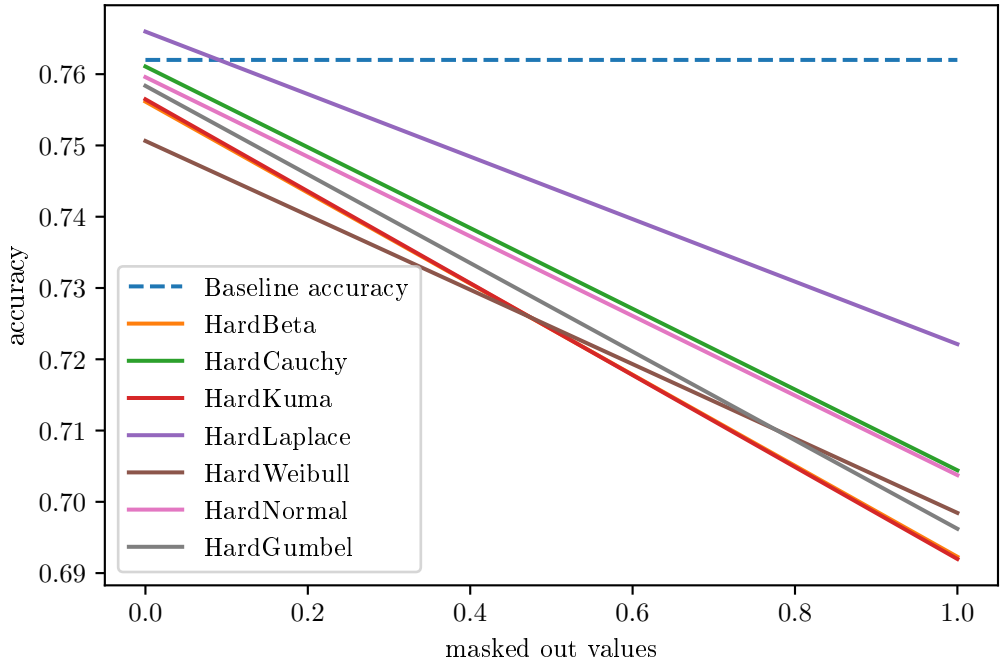


Figure 6: Linear approximation of the expected accuracy over mask out percentage for the PubMed dataset

Figures 4 to 6 compare the performance on each node classification dataset. We see that for each of those datasets masking none or only very few edges, but still weighting the edges, gives a small boost of performance to the overall model. We can also already observe that the choice of distribution is non-trivial and can have a big impact on the performance of the model after training.
The HardBeta, HardKuma, and HardNormal distributions all seem to be solid choices for the Cora (Figure 4) and CiteSeer (Figure 5) datasets, with the HardNormal distribution leading, especially on the CiteSeer dataset, while the HardLaplace distribution seems to perform by far the worst out of all distributions we tested. Here, the models using the top-performing distributions reliably outperform the baseline model, when no or only very few edges are removed. This is the consequence of a modified loss surface at training time, even though the KEdge layer does not mask out any edges at test time, generating only strictly positive weights. On the Pubmed dataset (Figure 6), this picture is distinctly different, as the HardLaplace distribution performs best at all levels of removed edges, and the HardBeta and HardKuma distributions seem to be two of the worst distributions to pick. The performance of all models, except the HardLaplace models at no masked edges, is worse than that of the baseline.

Graph classification

When training the models for the graph classification datasets, the first difference that stands out is, that our models seem to only attain certain levels of removed edges. During the training of these, the maskedness parameter λ did not control the percentage of removed edges, but more so the probability, that a model's percentage of removed edges reaches a given level. For the MUTAG dataset, almost all of the models we trained had a percentage of removed edges of 0, 0.07, 0.15, 0.75, 0.83, or 1 plus or minus 0.02. The levels of 0% and 100% were by far the most common ones we encountered. These set levels could be caused either by the shape of the loss surface, making it much more likely to end up near one of those levels, or they could be caused

by the datasets having only a relatively low number of node features, meaning that the attention mechanism in KEdge, that is responsible for masking out edges, can only differentiate a few classes of edges and either shows all of them or masks all of them at once.

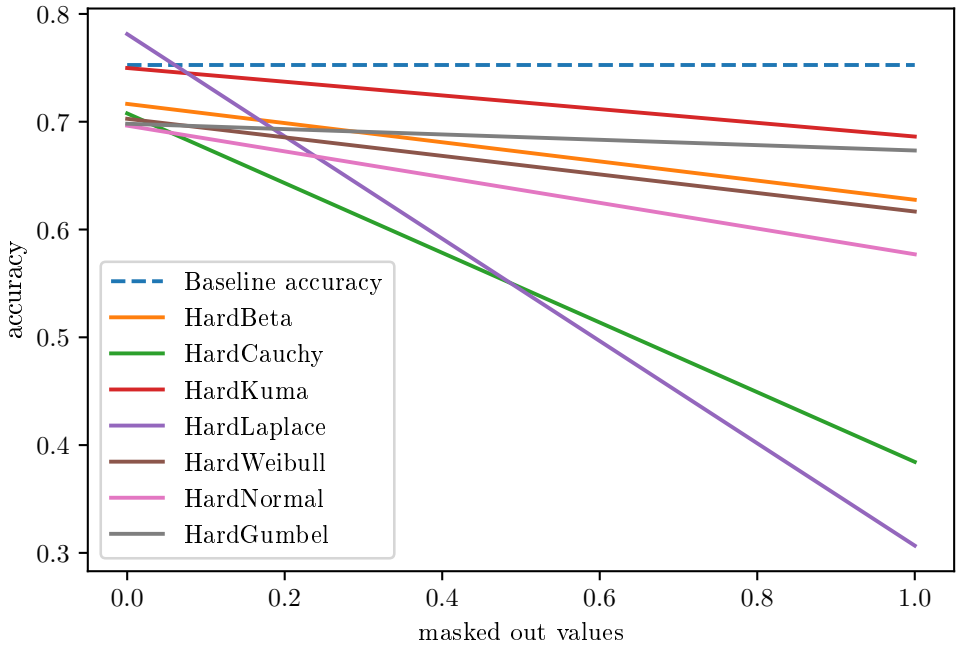


Figure 7: Linear approximation of the expected accuracy over mask out percentage for the MUTAG dataset, using global average pooling.

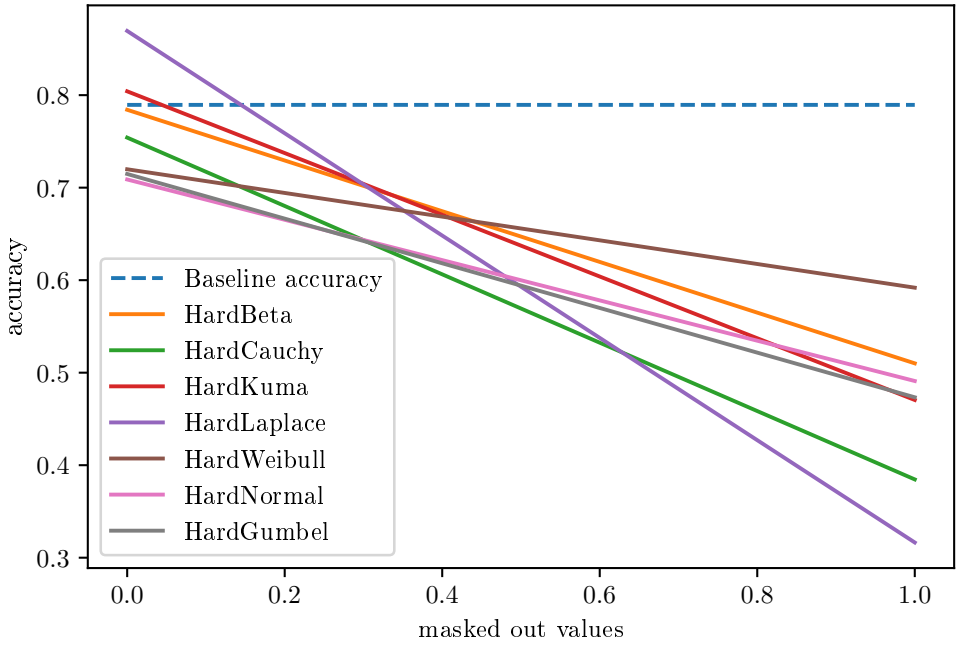


Figure 8: Linear approximation of the expected accuracy over mask out percentage for the MUTAG dataset, using global attention pooling.

In graph classification, we observed, that while the overall values might be different, the order of the distributions stays pretty much the same, no matter what kind of pooling we use. When using both GAP (Figure 7) and GAT (Figure 8) on the MUTAG dataset, the HardLaplace distribution offers the best performance out of all the distributions. It also is the only distribution, for which the model performs significantly better than the pure baseline model. When we increased the percentage of removed edges, the prediction of the models also dropped off rapidly and to way lower values than for the node classification datasets. This lets us conclude that for our graph classification datasets the connections in the graphs are more important than the actual nodes, while for the node classification tasks the node features are the more crucial of the two. In the raw data, we also observe, that when using attention pooling, some models attain a very low accuracy of about 0.3. Those outliers are not present when we use average pooling.
For a broad answer to Research Question 1, we would suggest to only increase the maskedness parameter λ with great caution, since the performance decreases from having more edges removed can be drastic. In terms of distributions, we also suggest one chooses the HardKuma or HardBeta distributions for a good baseline accuracy. For a better performing model, the HardNormal and HardLaplace distributions should also be considered.

4.2 Evaluation of Attribution Techniques

For our second experiment, we wanted to evaluate the quality of the explanations we gathered. The question to answer with this experiment is:

Research Question 2:
Under which circumstances is it better to first train a model and then equip it with one of the gradient-based, post-hoc attribution methods, and when to use a more interpretable model, equipped with KEdge, from the start? If one decides to use a gradient-based method, which one is of the most use to explain the model? On the other hand if one wants to use KEdge for interpretability, which distribution should one choose, and what about the maskedness parameter? Are these the same distributions that lead to a good model performance?

Experiment

To evaluate the gradient-based methods, we first trained the baseline model; that is the same model as described in Tables 3 and 4, but without the KEdge layer. After training, we used all suitable attribution methods from Section 2.2 on the model, to generate the attribution weights. These are Gradients, GradInput, SmoothGrad, and IntegratedGradients (IG), which are suitable for all models, as well as CAM, which is suitable for graph classification models using global average pooling (GAP) and GradCAM, which is suited for graph classification models, that use a pooling layer that is different from average pooling; in our case global attention pooling (GAT).
For the evaluation of KEdge, we trained the appropriate models and then used the methods of Section 3.2 to calculate the node level attention weights. We carried out this experiment using the sum, mean and max methods, but our results showed that using the mean yields the best results in 82% of all cases, as mentioned in Remark 3.2.1.
To evaluate the soft attribution weights, we used the integrated fidelity score (Definition 3.3.2). Our results are summarized in Tables 5 and 6. Empty spaces in Table 6 indicate, that the relevant level of removed edges could not be attained during training. For KEdge we only reported the results, using the mean to convert the edge weights to node weights, because of the aforementioned reason.

Results

At first, we can see from Table 5, that out of the gradient-based methods, the Integrated Gradients seem to consistently yield one of the highest integrated fidelity scores, no matter the dataset, followed by GradInput. When considering a gradient-based technique, these two should therefore be the go-to ones.

Method	Cora	CiteSeer	PubMed	MUTAG + GAP	MUTAG + GAT
Grads	0.815	0.727	0.906	0.56	0.74
GradInput	0.830	0.736	0.911	**0.61**	**0.76**
SmoothGrad	0.830	0.734	0.905	0.59	0.75
IG	**0.833**	**0.739**	**0.912**	0.60	**0.76**
CAM	-	-	-	0.55	-
GradCAM	-	-	-	-	0.72

Table 5: Fidelity of gradient based methods.

Node classification

Our results show, that for the node classification task, the fidelity of the weights from KEdge (Table 6) is generally lower than for the gradient-based methods (Table 5). This is expected, as KEdge acts as a global explanation on the whole dataset, while the gradient-based methods are all local explanations. As such, they tend to explain the influence of the input on the class of a single output node better. Here, the HardKuma, HardNormal, and HardGumbel distributions tend to produce the best explanations, with the HardLaplace distribution performing as one of the worst, consistently. The HardNormal and the HardKuma distributions also tend to yield good performing models, while the HardLaplace distribution tends to be either superb or awful. The fidelity scores of KEdge also decrease with the percentage of removed edges increasing. This seems to be counterintuitive at first, as one might expect the fidelity score to rise at first, since the least important edges are removed, leaving only the edges of higher importance. Nodes with a lot of edges of low importance should then have a lower attention weight, which should lead to a high fidelity score. So, with a rising percentage of removed edges, the attribution weights should at first be more expressive. We see in the data, that this is not at all the case, as every fidelity score decreases from 0% to 15% of removed edges. One possible explanation might be, that this effect mainly stems from the model performing worse when more edges are removed, as we saw in the first experiment (Section 4.1), and that worse performing models are generally harder to explain, as their outputs tend to be more random and less based on just a few important nodes.

Graph classification

In the case of graph classification, KEdge acts as a local explanation and yields higher fidelity scores than all the gradient-based explanations, regardless of distribution. In contrast to the node classification, here the integrated fidelity scores don't strictly decrease as the percentage of masked out edges increases, even though the model performance also decreases. Instead, we see a trend of scores, that decrease at first, as the percentage of masked edges increases, but after that starts to increase again, towards the 100% of masked out edges. Most of the time, the models in which all edges are masked out are even more interpretable than the models in which no edge was masked out. It could be the case that, in contrast to the node classification models that we suspected to become more random, these models actually need a lot of the edges in the graphs to make a reliable prediction, and once those edges are not present anymore they must retreat to more and more biased predictions. Such a biased model then gives a high fidelity score, since its prediction doesn't change much, no matter the input. This hypothesis is supported by

Dataset	% Rem. ±2%	KEdge + Kuma	KEdge + HardKuma	KEdge + HardCauchy	KEdge + HardGumbel	KEdge + HardLaplace	KEdge + HardNormal	KEdge + HardBeta	KEdge + HardWeibull
Cora	0%	0.67	**0.68**	0.62	0.67	0.58	**0.68**	0.67	0.65
	20%	-	0.66	0.57		0.55	0.64	0.65	0.62
	50%	-	0.59	0.54	0.56	0.53	0.59	0.58	0.57
	80%	-	0.54	0.53	0.51	0.52	0.53	0.53	0.54
	100%	-	0.53	0.53	0.54	0.52	0.53	0.52	0.53
CiteSeer	0%	0.61	**0.62**	0.55	0.61	0.53	0.61	0.59	0.57
	20%	-	0.59	0.53	0.59	0.51	0.57	0.58	0.56
	50%	-	0.54	0.51	0.54	0.50	0.55	0.55	0.54
	80%	-	0.52	0.49	0.51	0.50	0.53	0.53	0.51
	100%	-	0.51	0.49	0.52	0.50	0.52	0.52	0.51
PubMed	0%	**0.78**	0.77	0.77	**0.78**	0.75	**0.78**	0.77	0.74
	20%	-	0.75	0.76	0.74	0.73	0.76	0.74	0.72
	50%	-	0.70	0.71	0.68	0.71	0.72	0.71	0.69
	80%	-	0.67	0.67	0.65	0.68	0.67	0.68	0.66
	100%	-	0.65	0.66	0.65	0.66	0.65	0.65	0.65
MUTAG + GAP	0%	0.67	0.87	0.75	0.90	0.63	0.87	0.81	**1.0**
	15%	-	0.76	0.89	0.89	0.7	0.93	0.73	0.81
	85%	-	0.77	0.79				0.99	0.67
	100%	-	0.91	0.88	0.95	**1.0**	**1.0**	0.92	0.93
MUTAG + GAT	0%	0.77	0.85	0.86	0.90	0.76	0.94	0.83	0.88
	15%	-	0.79	0.8	0.84	0.77	0.78	0.80	0.75
	85%	-	0.73	0.83	**1.0**			0.79	**1.0**
	100%	-	0.94	0.92	0.93	**1.0**	0.97	0.80	0.93

Table 6: Integrated Fidelity of Gradient Based Methods and KEdge.

our first experiment, where we saw that for the graph classification task, the edges are of crucial importance. So, removing all edges probably leads to a highly biased, low-performing model with only little regard for the actual input.
When comparing the different distributions for KEdge in Table 6, we see that at the level of no masked edges the HardWeibull and HardGumbel distributions perform the best, followed by the HardNormal distribution. The HardLaplace distribution, which had the most accurate models also has the lowest fidelity scores, while the HardWeibull and HardGumbel distributions lead to relatively poor model performances.
Now, we can derive an answer to Research Question 2. For node classification, one should use gradient-based methods, especially integrated gradients or GradInput, when there is no special interest in global explanations, as they tend to better explain the model's decision for single nodes. For the graph classification task, this picture is inverted and the KEdge methods outperform the gradient-based ones. It again makes the most sense to set the maskedness to $\lambda = 0$ as to not compromise the model's performance. This does not mean, that no edges are masked, but that only edges are masked, such that the expected model performance does not decrease, as the regularization on Z in Equation (3) is omitted. Additionally, KEdge can introduce positive weights by choosing the expected value when the point probabilities at zero and one become sufficiently small. For a choice of distribution, we would suggest the HardNormal, HardKuma, or HardBeta distributions, depending on which one offers the best performance on the particular dataset.

4.3 Explaining Movie Reviews

In our third and final experiment, we again wanted to judge the quality of explanations of the different methods, by comparing them to human explanations. The question we want to answer is:

Research Question 3:
Which method offers good, concise explanations of model predictions? And, by which metric should one judge attribution weights?

Since explaining decisions, and their contributing factors, for graphs, is hard, even for experts, we went to the domain of text. Here, we used the annotated movie reviews dataset from DeYoung et al. to compare our attribution weights to human explanations.

Experiment

The first step to conduct this experiment is to convert the text of a movie review into a graph so that we can apply graph neural networks, and KEdge in particular. Here, we employed two Graph-of-Words [RV13] strategies that led to two datasets.
For both datasets, we first went over the text and removed a few non-informative terms like punctuation marks and the most common words[4]. For an example, see the differences from Figure 9 to Figure 10.

> ok , i admit it – i find camp amusement with the spice girls . yes , the same spice girls of the gimmicky individual " identities , " they of the cheesy unifying mantra of " girls power . "

Figure 9: Raw movie review example text.

We then treated all remaining words as nodes in a graph. It is here, where both datasets differ. For the so-called linear dataset, we let every occurrence of a word be a unique node (see Figure 11), while for the complex dataset, each occurrence of a word corresponds to the same, single node in the graph (see Figure 12).

[4]We removed all of those tokens: [',', '.', "s', '"', '-', '?', '!', '/', '(', ')', '_', 'the', 'be', 'to', 'of', 'and', 'a', 'an', 'in', 'that', 'it', 'you', 'me', 'i', 'is', 'at'].

ok admit find camp amusement with spice girls yes same spice girls gimmicky individual identities they cheesy unifying mantra girls power

Figure 10: Movie review example text with non-informative tokens removed.

For the edges of the graph, we took a sliding window of size three and connected the nodes corresponding to the words inside the window. The resulting graph is undirected and has no edge weights. If an edge was already present in the graph, we didn't add it a second time. So, no multi-edges are allowed. As the last step, we transformed the words that correspond to the nodes in the graph to 300 dimensional node features using GloVe [PSM14] word embeddings[5].

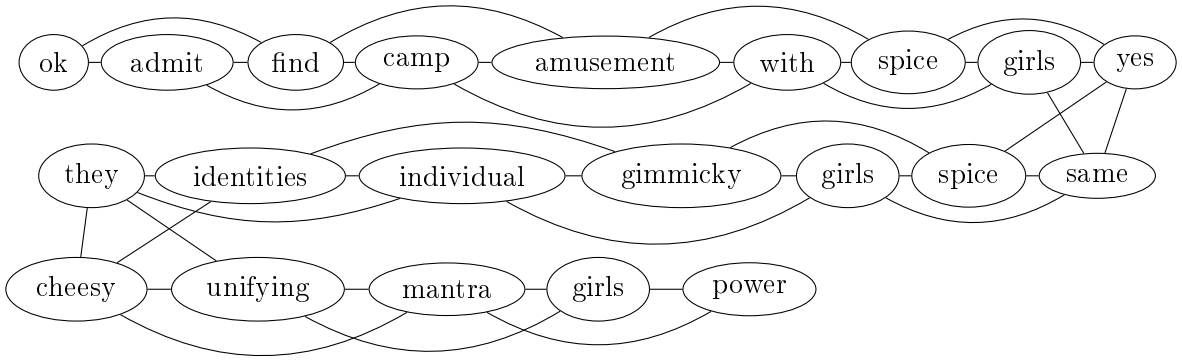


Figure 11: Movie review example; linear graph.

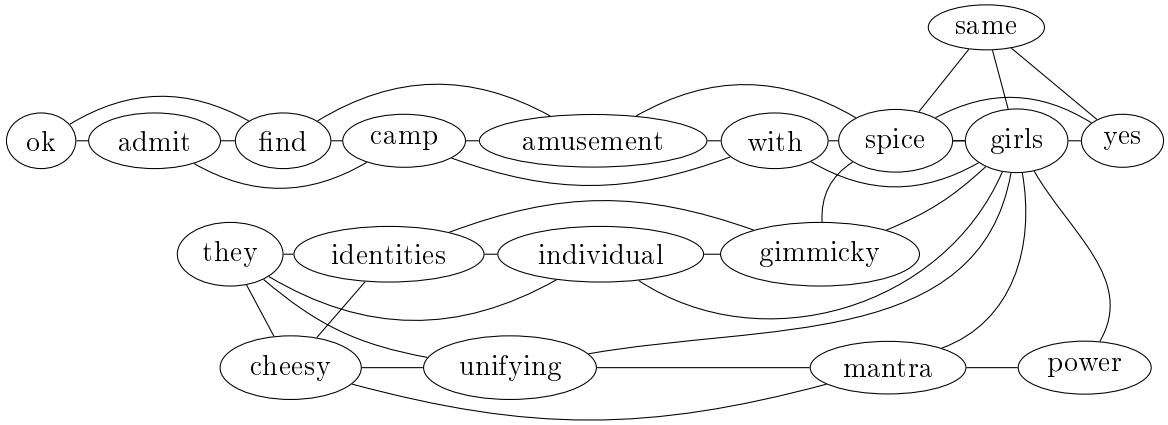


Figure 12: Movie review example; complex graph.

For both datasets, the model architecture, described in Table 7, was the same. It again is made up of two GNN layers and a pooling layer, but this time followed by a more complex network, rather than just a single linear layer like in the first experiments (Table 4). We chose a maskedness parameter of $\lambda = 0$ for training.

After training the models for all possible combinations of the baseline model (no KEdge) or KEdge with all distributions, linear or complex dataset and average pooling or attention pooling, we evaluated the model accuracy and the fraction of removed edges, as well as the AUPRC, the Comprehensiveness AOPC, the Sufficiency AOPC[6], and the integrated fidelity from Section 3.3.

[5]Specifically the *GloVe 840B 300d* embeddings.

[6]We calculate the AUPRC, Comprehensiveness AOPC, and Sufficiency AOPC using the implementation from DeYoung et al.

Operation	Feature size	Weights
Input	300	
KEdge		$2 \cdot (300 \cdot 32 + 2 \cdot 32)$
GraphConv		$2 \cdot 300 \cdot 300 + 300$
LeakyReLU	300	
GraphConv		$2 \cdot 300 \cdot 200 + 200$
LeakyReLU	200	
pooling		GAP: 0 / GAT: $200 \cdot 5 + 5 + 5 \cdot 1 + 1$
Linear		$200 \cdot 100 + 100$
LeakyReLU + BatchNorm	100	$1 + 100$
Linear		$100 \cdot 10 + 10$
LeakyReLU + BatchNorm	10	$1 + 10$
Linear		$10 \cdot 5 + 5$
LeakyReLU + BatchNorm	5	$1 + 5$
Linear		$5 \cdot 2 + 1$
Softmax	2	

Table 7: Model architecture for movies dataset.

Results

Our first observation was that the linear dataset overwhelmingly performs better when using average pooling, while for the complex dataset results are more mixed, but more often than not attention pooling performs better than average pooling. Here, we just report the results for the linear dataset with average pooling (Table 8) and for the complex dataset with attention pooling (Table 9). For a full list of all results, see Appendix A.

Model	acc	rem.	AUPRC	Comp. AOPC	Suff. AOPC	int. fid.
KEdge + HardBeta	0.869	0.768	0.368	0.280	0.199	0.900
KEdge + HardCauchy	0.864	0.019	0.353	0.206	0.154	0.897
KEdge + HardGumbel	0.879	0.421	0.417	0.167	0.279	0.904
KEdge + HardKuma	0.884	0.755	0.392	0.287	0.150	0.928
KEdge + HardLaplace	0.869	0.076	0.362	0.213	0.169	0.908
KEdge + HardWeibull	0.874	0.563	0.402	0.231	0.181	0.873
KEdge + HardNormal	**0.889**	**0.831**	0.403	0.195	0.172	0.936
KEdge + Kuma	0.869	0.000	**0.547**	0.068	0.250	0.888
Baseline + CAM	0.864	-	0.389	**0.394**	0.190	0.927
Baseline + GradInput	0.864	-	0.386	0.377	0.207	0.914
Baseline + Grads	0.864	-	0.396	0.359	0.240	0.907
Baseline + IG	0.864	-	0.397	**0.394**	**0.120**	**0.938**
Baseline + SmoothGrad	0.864	-	0.386	0.313	0.230	0.920
Baseline + Random	0.864	-	0.547	0.074	0.241	0.852

Table 8: Movie Reviews for linear dataset & GAP.

For the linear dataset with average pooling, the best performing model was the one that uses KEdge with the HardNormal distribution at 89% accuracy, followed by the HardKuma and HardGumbel distributions at 88%, against a baseline of 86% accuracy. On this dataset, the accuracy and percentage of removed edges have a positive correlation of 0.56. This is unlike the datasets we looked at in the first experiment (Section 4.1) and signifies that the linear graph has some edges, that hurt the model's performance.
When looking at the metrics for the attribution weights, we saw that the baseline model with integrated gradients is performing the best across Comprehensiveness AOPC, Sufficiency AOPC and also integrated fidelity, but not AUPRC. While the gradient-based methods all had an

AUPRC of around 0.39, the KEdge methods ranged from about 0.35 to about 0.41, with one big outlier: KEdge with the Kuma distribution reached an AUPRC of 0.547, while also attaining the overall lowest Comprehensiveness AOPC of 0.068 and the second-highest Sufficiency AOPC of all models on the linear dataset, using average pooling, at 0.250. All these, however, are the result of constant attribution weights of 1! We also trained a baseline model and manually gave it attribution weights of 1 and that model reached the same AUPRC of 0.547, a Comprehensiveness AOPC of 0.074, and a Sufficiency AOPC of 0.241.
Compared to the models on the ERASER leaderboard, only our random models beat the highest AUPRC of 0.502 and no model beat the lowest Sufficiency AOPC of 0.093, while almost all of our models with actual, informational attention weights, so not the Random baseline and the Kuma model, beat the top Comprehensiveness AOPC of 0.187.

Model	acc	rem.	AUPRC	Comp. AOPC	Suff. AOPC	int. fid.
KEdge + HardBeta	0.859	0.373	0.389	0.150	0.199	0.887
KEdge + HardCauchy	**0.889**	0.020	0.374	0.251	0.224	0.875
KEdge + HardGumbel	0.864	0.790	0.350	0.194	**0.159**	0.912
KEdge + HardKuma	0.844	0.566	0.362	0.202	0.184	0.884
KEdge + HardLaplace	0.854	0.070	0.368	0.241	0.300	0.901
KEdge + HardWeibull	0.859	**0.858**	0.325	**0.296**	0.312	0.889
KEdge + HardNormal	0.864	0.613	0.369	0.263	0.243	0.875
KEdge + Kuma	0.854	0.000	**0.547**	0.123	0.292	0.774
Baseline + GradCAM	0.864	-	0.362	0.163	0.293	0.857
Baseline + GradInput	0.864	-	0.358	0.179	0.226	0.924
Baseline + Grads	0.864	-	0.365	0.167	0.258	0.923
Baseline + IG	0.864	-	0.346	0.143	0.278	0.926
Baseline + SmoothGrad	0.864	-	0.358	0.162	0.229	**0.927**
Baseline + Random	0.864	-	0.547	0.067	0.251	0.855

Table 9: Movie Reviews for complex dataset & GAT.

On the complex dataset with attention pooling, the best performing models were the KEdge model using the HardCauchy distribution, which reaches an accuracy of 89%, followed by the HardNormal and HardGumbel distributions with 86%. Therefore, almost all KEdge models performed worse than the 86% accuracy baseline model. Unlike on the linear dataset, the accuracy and percentage of removed edges are almost uncorrelated (correlation of 0.11), implying that the edges of the complex graphs are more important for a good performing model than on the linear dataset; or at least they are not as obstructive.
Again, the KEdge model with the Kuma distribution archived the highest AUPRC score of 0.547. The largest Comprehensiveness AOPC was obtained by the HardWeibull model at 0.296. Overall only the model using the HardGumbel distribution and average pooling on the complex dataset, as well as some of the baseline models on the linear dataset with GAP, beat this score. It also had more than one and a half times the Comprehensiveness AOPC of the best model on the ERASER leaderboard, which stands at 0.187.
The highest fidelity score is attained by the baseline model using SmoothGrad, followed by the same model using the integrated gradients method. We also see, that the gradient-based methods generally score better at integrated fidelity than the KEdge models do on this dataset.

	AUPRC	C. AOPC	S. AOPC	int. fid.
AUPRC	1.0	-0.333	-0.064	-0.209
C. AOPC	-0.333	1.0	-0.344	0.564
S. AOPC	-0.064	-0.344	1.0	-0.438
int. fid.	-0.209	0.554	-0.438	1.0

Table 10: Correlation of different metrics on movie dataset.

When comparing the four metrics, it stands out that they are only mildly correlated (see Table 10). While the S. AOPC and C. AOPC are mildly negatively correlated, which is expected since we want to attain a high C. AOPC, but a low S. AOPC, the S. AOPC, and the AUPRC are almost uncorrelated, while AUPRC and Comprehensiveness AOPC are even negatively correlated.
The AUPRC and integrated fidelity scores are only very mildly negatively correlated, while integrated fidelity has a relatively large positive correlation with the Comprehensiveness AOPC and a negative one with the Sufficiency AOPC. This means that when one of those scores gets better, the fidelity score tends to also get better, which is to be expected.
This suggests that the attributions, that explain the model tend to be independent of human rationales or even that making rationales more like human explanations worsens the rationales' abilities to explain a neural model. And this suggests, that neural models either just work differently to human brains and therefore focus on different parts of the text or that human rationales are not that good at explaining human decisions.
Now, to get a better feel for the attributions and the metrics, we considered a few examples of movie reviews with visualized attribution scores. The visualizations are all normalized, such that the lowest weight and highest weight for each model are mapped to the same color each time (see Figure 13). We did this for a selected range of models.

Figure 13: Scale of the visualization; low to high.

Out of all models on the linear dataset with average pooling, we took the HardKuma model, because it has the highest Comprehensiveness AOPC and lowest Sufficiency AOPC out of all the KEdge models, we chose the HardNormal model, because it has the highest accuracy, and we chose the baseline model, using integrated gradients since it has the highest integrated fidelity score, as well as the best Sufficiency and Comprehensiveness AOPC. We also took the HardGumbel model, since it has the highest Sufficiency AOPC, which should mean that attributions are bad here. For models on the complex dataset with attention pooling, we took the HardCauchy one, again, because it performs the best, we took the HardWeibull model, because it has the highest Comprehensiveness AOPC, and we chose the SmoothGrad one because it has the highest fidelity score. We also took some of the other models (see Appendix A). From the linear dataset with attention pooling we took the HardWeibull model because it performs the best overall, we took the HardKuma model, because of its AUPRC of 0.516 and good performance and we took the HardBeta model for having the overall worst AUPRC. From the models on the complex dataset with average pooling, we only chose the HardWeibull model, because it has the overall lowest integrated fidelity. For convenience, we have labeled the models.
We compared the attributions on small parts of three movie reviews, that are marked as important by the human rationales, while also being important to at least some of the models. The results can be seen in Table 11. For the full reviews and ground truth, human attributions see Appendix B.
The first thing that stands out about the attributions is, how different they are. They range from no real weight on important parts of the text to the whole thing being bright red. Upon further inspection of the full reviews for models 5, 6, 7, 10 and 11 one finds that they disproportionally value words, that occur often, even if they do not contribute to the interpretation of the review. Examples of this are the word 'movie', the title of the film, or the name of an actor. Then there is model 9, which weighs most of the words very highly with only a few exceptions. This case only occurs for KEdge models, the reason being that they simply only mask out very few edges. This is the same dilemma we found in the second experiment (Section 4.2), where more masked out edges would, in principle, lead to more clear attributions, but also worse performance, making the attributions worse again. This model was also chosen, because of its high AUPRC, which turned out to be a fluke, like the Kuma models we saw before.

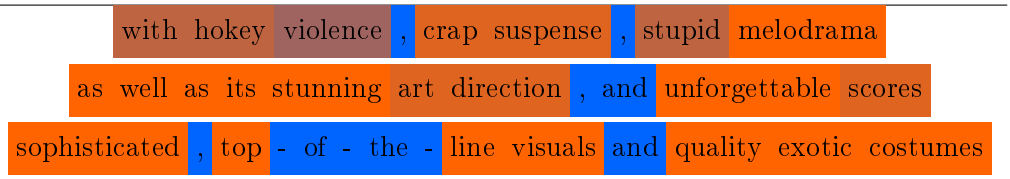
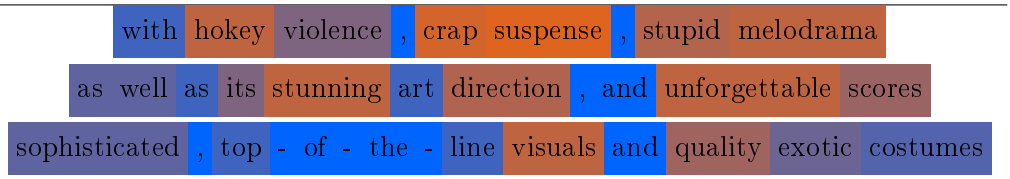
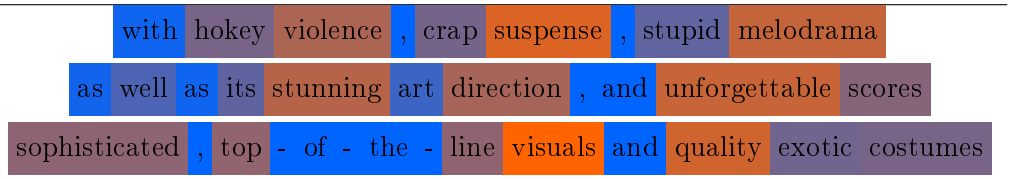
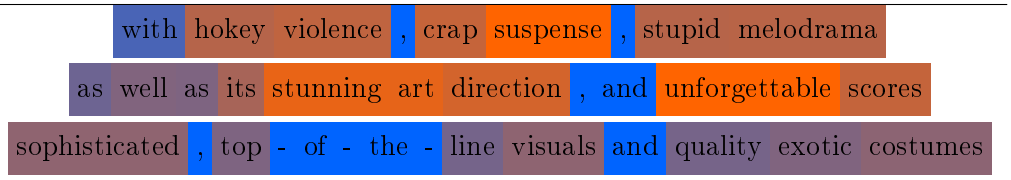
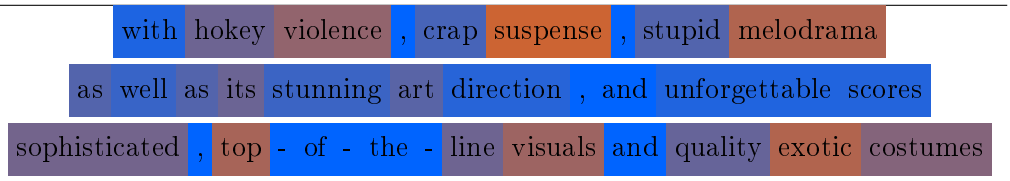
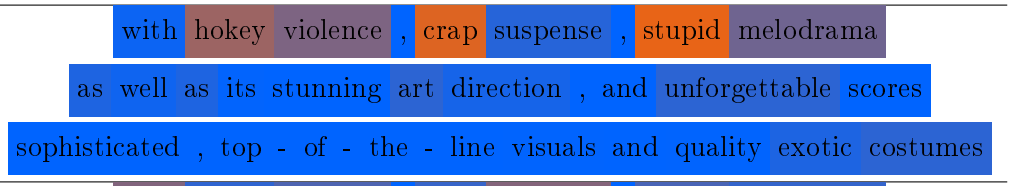
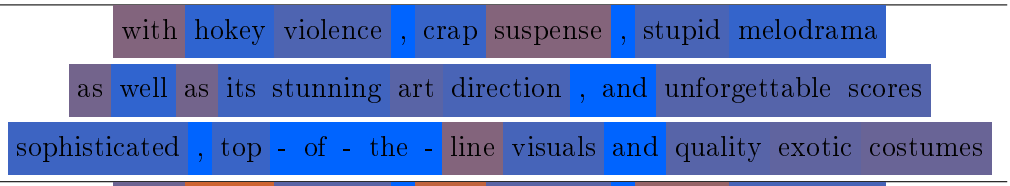
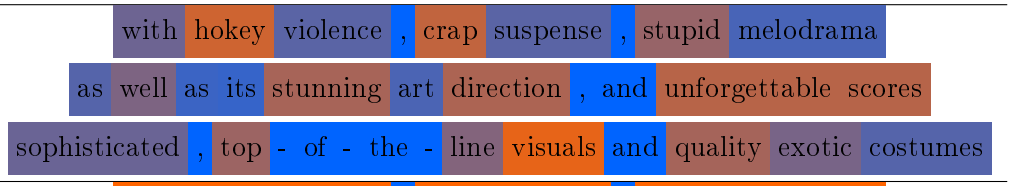
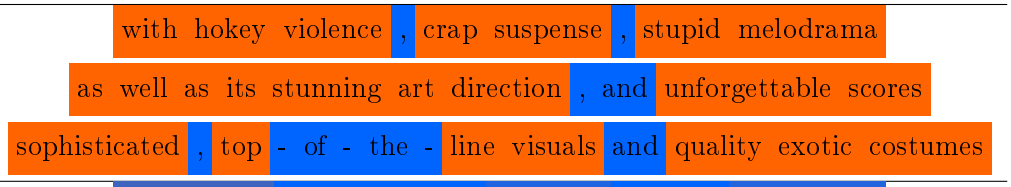
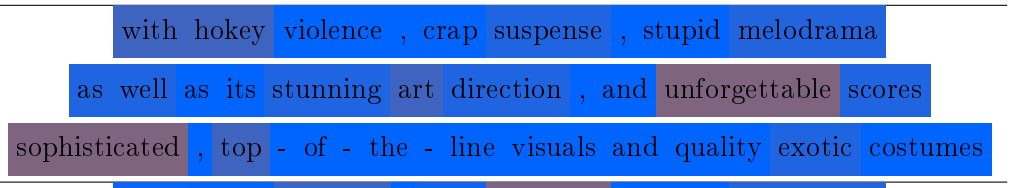
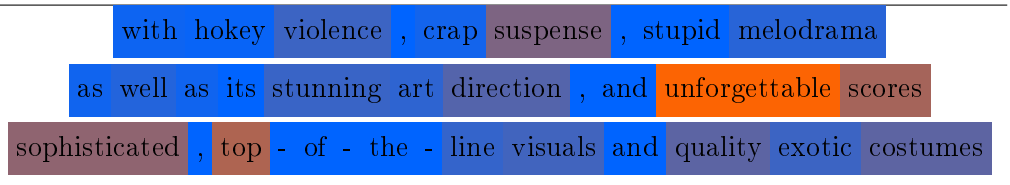
No.	Model	Example attributions
1	KEdge + HardGumbel linear & GAP	
2	KEdge + HardKuma linear & GAP	
3	KEdge + HardNormal linear & GAP	
4	Baseline + IG linear & GAP	
5	KEdge + HardCauchy complex & GAT	
6	KEdge + HardWeibull complex & GAT	
7	Baseline + SmoothGrad complex & GAT	
8	KEdge + HardWeibull linear & GAT	
9	KEdge + HardKuma linear & GAT	
10	KEdge + HardBeta linear & GAT	
11	KEdge + HardWeibull complex & GAP	

Table 11: Attribution of significant parts of three reviews; selected models.

From the examples, models number 2 to 4 and 8 look the best to the human eye. Here the more important words are clearly differentiated, with the qualifiers, that really convey the authors' opinion, weighted even more than the nouns they reference ('hokey violence', 'crap suspense', 'stunning art direction'). Model 4 stands out for the fact, that the words with the highest weights are not overshadowed by other words in the reviews, outside of the excerpts, like it is the case for models 2, 3, and even more so for model 8.
Models 2 and 4 were explicitly chosen for the high integrated fidelity, high Comprehensiveness AOPC and low Sufficiency AOPC, while models 3 and 8 were chosen for their good performance. Model number 7 was also chosen because of its high integrated fidelity score, but it doesn't stand up to the eye-test as 3 and 4 do.
The contrast between models 9 and 10 is especially large, considering the fact, that both were chosen because of their AUPRC scores. Model 9 had the highest while model 10 had the lowest AUPRC, yet the explanations of model 10 carry at least some information on what words are important for a decision, while this is not the case for model 10.
Aside from that, there is not a distinct best metric among integrated fidelity, Comprehensiveness AOPC, and Sufficiency AOPC. While all of them can be indicators for good-looking attributions, there are also some outliers for all of them. That being said, multiple metrics on a high level seem to be a sign for good explanations.
Now, to answer Research Question 3, it can be said, that while using KEdge over gradient-based methods can lead to very low-informational attributions like it's the case for model 9, this does not happen very often, when choosing different distributions. We also saw, that when considering Graph-of-Words techniques, the linear graphs tend to be more explainable. As for metrics, comparison to human explanations and especially AUPRC is a bad way of judging explanations for a GNN model, while, at least in the domain of text, the other metrics we considered are all indicators, but not assurances of good looking explanations.

5 Conclusion

Since graph neural networks are on the rise and used in high-stakes environments, it's more important than ever before to explain the decisions made by them. We studied a variety of different explanation techniques and investigated when to use each one, based on model performance and our new metric of integrated fidelity. We saw that for node classification one probably wants to use the integrated gradients as the most solid of the gradient-based techniques unless global explanations are specifically required. In this case, KEdge should be used. We also saw that KEdge outperforms the gradient-based techniques for graph classification. When using KEdge, we recommend using the HardKuma, HardBeta, or HardNormal distributions, as all of them offer accurate models, as well as high fidelity explanations. When the model's accuracy is the top priority, one should also try the HardLaplace distribution, as it can outperform the aforementioned distributions, depending on the dataset that used. However, its explanations are lower in fidelity. We also learned, that when using a Graph-of-Words approach to tackle a problem in the text domain, the linear type of graph lends itself to models, which can be explained better, while not sacrificing any model performance, and that explanations, which mimic human rationales are bad at explaining the decisions of GNNs.
Further research into interpretable by design GNN layers could shine a light onto an algorithm similar to KEdge, but focused on nodes. This would remove the intermediate step of converting edge weights to node weights. One could use the node features to calculate the parameters of one of the hard distributions we introduced to then sample some weight for each node. The node's features could then be multiplied by the sampled weight. To completely remove a node, that is weighted with zero, more complex changes to the resulting graph and also the GNN itself have to be made. Another way attribution might be possible, at least for graph classification, is to simply use the weights generated by an attention pooling layer. Here further research is needed to answer the question if that approach is any good, or if it even is better than the attribution techniques this thesis focused on. Since our comparison of metrics for attribution weights yielded no one, best metric, further research is needed into which metric for judging attribution weights is best for a given task and focus.

References

[BBS21] Pietro Bongini, Monica Bianchini, and Franco Scarselli. "Molecular generative Graph Neural Networks for Drug Discovery". In: *Neurocomputing* 450 (2021), pp. 242–252. DOI: 10.1016/j.neucom.2021.04.039.

[BMR21] Alaa Bessadok, Mohamed Ali Mahjoub, and Islem Rekik. *Graph Neural Networks in Network Neuroscience*. 2021. arXiv: 2106.03535 [cs.LG].

[Dek+05] Frederik Michel Dekking et al. "Testing hypotheses: essentials". In: *A Modern Introduction to Probability and Statistics: Understanding Why and How*. London: Springer London, 2005, pp. 373–382. ISBN: 978-1-84628-168-6. DOI: 10.1007/1-84628-168-7_25.

[DeY+20] Jay DeYoung et al. *ERASER: A Benchmark to Evaluate Rationalized NLP Models*. 2020. arXiv: 1911.03429 [cs.CL].

[FKA21] Thorben Funke, Megha Khosla, and Avishek Anand. *Zorro: Valid, Sparse, and Stable Explanations in Graph Neural Networks*. 2021. arXiv: 2105.08621 [cs.LG].

[FL19] Matthias Fey and Jan Eric Lenssen. *Fast Graph Representation Learning with PyTorch Geometric*. 2019. arXiv: 1903.02428 [cs.LG].

[GF17] Bryce Goodman and Seth Flaxman. "European Union Regulations on Algorithmic Decision-Making and a "Right to Explanation"". In: *AI Magazine* 38 (2017), pp. 50–57. DOI: 10.1609/aimag.v38i3.2741.

[Hu+21] Weihua Hu et al. *Open Graph Benchmark: Datasets for Machine Learning on Graphs*. 2021. arXiv: 2005.00687 [cs.LG].

[Kap+20] Amol Kapoor et al. *Examining COVID-19 Forecasting using Spatio-Temporal Graph Neural Networks*. 2020. arXiv: 2007.03113.

[KW17] Thomas N. Kipf and Max Welling. *Semi-Supervised Classification with Graph Convolutional Networks*. 2017. arXiv: 1609.02907 [cs.LG].

[LAS20] Hima Lakkaraju, Julius Adebayo, and Sameer Singh. *Explaining Machine Learning Predictions: State-of-the-art, Challenges, and Opportunities*. NeurIPS 2020. Dec. 7, 2020. URL: explainml-tutorial.github.io (visited on 06/22/2021).

[Li+17] Yujia Li et al. *Gated Graph Sequence Neural Networks*. 2017. arXiv: 1511.05493 [cs.LG].

[Mor+20] Christopher Morris et al. *Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks*. 2020. arXiv: 1810.02244 [cs.LG].

[Pas+19] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035.

[Pen+18] Hao Peng et al. "Large-Scale Hierarchical Text Classification with Recursively Regularized Deep Graph-CNN". In: *Proceedings of the 2018 World Wide Web Conference*. Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee, 2018. DOI: 10.1145/3178876.3186005.

[PSM14] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. "GloVe: Global Vectors for Word Representation". In: *Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162.

[Rat+21] Mandeep Rathee et al. "Learnt Sparsification for Interpretable Graph Neural Networks". In: (2021). arXiv: 2106.12920 [cs.LG].

[Rud87] W. Rudin. *Real and Complex Analysis*. Mathematics series. McGraw-Hill, 1987.

[RV13] François Rousseau and Michalis Vazirgiannis. "Graph-of-word and TW-IDF: new approach to ad hoc IR". In: *International Conference on Information and Knowledge Management, Proceedings*. Oct. 2013, pp. 59–68. DOI: 10.1145/2505515.2505671.

[San+20] Benjamin Sanchez-Lengeling et al. "Evaluating Attribution for Graph Neural Networks". In: *Advances in Neural Information Processing Systems 33*. 2020.

[Sch+17] Michael Schlichtkrull et al. *Modeling Relational Data with Graph Convolutional Networks*. 2017. arXiv: 1703.06103 [stat.ML].

[Sel+19] Ramprasaath R. Selvaraju et al. "Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization". In: *International Journal of Computer Vision* 128.2 (Oct. 2019), pp. 336–359. DOI: 10.1007/s11263-019-01228-7.

[Shr+17] Avanti Shrikumar et al. *Not Just a Black Box: Learning Important Features Through Propagating Activation Differences*. 2017. arXiv: 1605.01713 [cs.LG].

[SK20] Zeren Shui and George Karypis. *Heterogeneous Molecular Graph Neural Networks for Predicting Molecule Properties*. 2020. arXiv: 2009.12710 [cs.LG].

[Smi+17] Daniel Smilkov et al. *SmoothGrad: removing noise by adding noise*. 2017. arXiv: 1706.03825 [cs.LG].

[STY17] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. *Axiomatic Attribution for Deep Networks*. 2017. arXiv: 1703.01365 [cs.LG].

[Uni16] European Union. *Regulation (EU) 2016/679 of the European Parliament and of the Council*. 2016. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A02016R0679-20160504&qid=1532348683434 (visited on 10/06/2021).

[Vel+18] Petar Veličković et al. *Graph Attention Networks*. 2018. arXiv: 1710.10903 [stat.ML].

[Wu+19] Felix Wu et al. *Simplifying Graph Convolutional Networks*. 2019. arXiv: 1902.07153 [cs.LG].

[YCS16] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. *Revisiting Semi-Supervised Learning with Graph Embeddings*. 2016. arXiv: 1603.08861 [cs.LG].

[Yin+18] Rex Ying et al. "Graph Convolutional Neural Networks for Web-Scale Recommender Systems". In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (July 2018). DOI: 10.1145/3219819.3219890.

[Zho+15] Bolei Zhou et al. *Learning Deep Features for Discriminative Localization*. 2015. arXiv: 1512.04150 [cs.CV].

[Zho+21] Jie Zhou et al. *Graph Neural Networks: A Review of Methods and Applications*. 2021. arXiv: 1812.08434 [cs.LG].

A Movie Reviews - Metrics

Model	Pooling	Dataset	acc	rem	AUPRC	Comp. AOPC	Suff. AOPC	int. fid.
KEdge + HardBeta	GAP	complex	0.879	0.739	0.349	0.252	0.163	0.881
KEdge + HardBeta	GAT	complex	0.859	0.373	0.389	0.150	0.199	0.887
KEdge + HardBeta	GAP	linear	0.869	0.768	0.368	0.280	0.199	0.900
KEdge + HardBeta	GAT	linear	0.879	0.897	0.300	0.158	0.222	0.845
KEdge + HardCauchy	GAP	complex	0.844	0.045	0.363	0.233	0.208	0.863
KEdge + HardCauchy	GAT	complex	0.889	0.020	0.374	0.251	0.224	0.875
KEdge + HardCauchy	GAP	linear	0.864	0.019	0.353	0.206	0.154	0.897
KEdge + HardCauchy	GAT	linear	0.879	0.084	0.359	0.172	0.373	0.782
KEdge + HardGumbel	GAP	complex	0.874	0.746	0.371	0.331	0.274	0.862
KEdge + HardGumbel	GAT	complex	0.864	0.790	0.350	0.194	0.159	0.912
KEdge + HardGumbel	GAP	linear	0.879	0.421	0.417	0.167	0.279	0.904
KEdge + HardGumbel	GAT	linear	0.894	0.739	0.343	0.175	0.159	0.904
KEdge + HardKuma	GAP	complex	0.864	0.804	0.380	0.232	0.188	0.912
KEdge + HardKuma	GAT	complex	0.844	0.566	0.362	0.202	0.184	0.884
KEdge + HardKuma	GAP	linear	0.884	0.755	0.392	0.287	0.150	0.928
KEdge + HardKuma	GAT	linear	0.889	0.058	0.516	0.113	0.223	0.870
KEdge + HardLaplace	GAP	complex	0.874	0.032	0.371	0.266	0.183	0.907
KEdge + HardLaplace	GAT	complex	0.854	0.070	0.368	0.241	0.300	0.901
KEdge + HardLaplace	GAP	linear	0.869	0.076	0.362	0.213	0.169	0.908
KEdge + HardLaplace	GAT	linear	0.859	0.173	0.366	0.332	0.208	0.926
KEdge + HardWeibull	GAP	complex	0.889	0.884	0.346	0.145	0.287	0.764
KEdge + HardWeibull	GAT	complex	0.859	0.858	0.325	0.296	0.312	0.889
KEdge + HardWeibull	GAP	linear	0.874	0.563	0.402	0.231	0.181	0.873
KEdge + HardWeibull	GAT	linear	0.899	0.588	0.378	0.217	0.178	0.923
KEdge + HardNormal	GAP	complex	0.874	0.956	0.337	0.183	0.191	0.855
KEdge + HardNormal	GAT	complex	0.864	0.613	0.369	0.263	0.243	0.875
KEdge + HardNormal	GAP	linear	0.889	0.831	0.403	0.195	0.172	0.936
KEdge + HardNormal	GAT	linear	0.874	0.597	0.389	0.207	0.205	0.888
KEdge + Kuma	GAP	complex	0.799	0.000	0.547	0.080	0.235	0.856
KEdge + Kuma	GAT	complex	0.854	0.000	0.547	0.123	0.292	0.774
KEdge + Kuma	GAP	linear	0.869	0.000	0.547	0.068	0.250	0.888
KEdge + Kuma	GAT	linear	0.829	0.000	0.547	0.070	0.259	0.849
Baseline + CAM	GAP	complex	0.859	-	0.365	0.276	0.218	0.909
Baseline + CAM	GAP	linear	0.864	-	0.389	0.394	0.190	0.927
Baseline + GradCAM	GAT	complex	0.864	-	0.362	0.163	0.293	0.857
Baseline + GradCAM	GAT	linear	0.874	-	0.419	0.249	0.214	0.839
Baseline + GradInput	GAP	complex	0.859	-	0.346	0.213	0.319	0.870
Baseline + GradInput	GAT	complex	0.864	-	0.358	0.179	0.226	0.924
Baseline + GradInput	GAP	linear	0.864	-	0.386	0.377	0.207	0.914
Baseline + GradInput	GAT	linear	0.874	-	0.417	0.216	0.165	0.873
Baseline + Grads	GAP	complex	0.859	-	0.350	0.198	0.349	0.859
Baseline + Grads	GAT	complex	0.864	-	0.365	0.167	0.258	0.923
Baseline + Grads	GAP	linear	0.864	-	0.396	0.359	0.240	0.907
Baseline + Grads	GAT	linear	0.874	-	0.425	0.212	0.182	0.876
Baseline + IG	GAP	complex	0.859	-	0.341	0.145	0.301	0.911
Baseline + IG	GAT	complex	0.864	-	0.346	0.143	0.278	0.926
Baseline + IG	GAP	linear	0.864	-	0.397	0.394	0.120	0.938
Baseline + IG	GAT	linear	0.874	-	0.414	0.211	0.177	0.873
Baseline + SmoothGrad	GAP	complex	0.859	-	0.344	0.159	0.342	0.880
Baseline + SmoothGrad	GAT	complex	0.864	-	0.358	0.162	0.229	0.927
Baseline + SmoothGrad	GAP	linear	0.864	-	0.386	0.313	0.230	0.920
Baseline + SmoothGrad	GAT	linear	0.874	-	0.418	0.204	0.173	0.866

Table 12: Eraser stats for different models.

B Example Movie Reviews

Table 13: Attribution weighted movie reviews. Model numbers are from Table 11.

Ground truth

Review 1

martial arts master steven seagal (not to mention director !) has built a career out of playing an allegedly fictitious martial arts superman who never gets hurt in fights , talks in a hushed tone , and squints at any sign of danger . he 's also the most consistent individual in hollywood today , since all his movies suck . they basically represent his egotisitical tendencies about his art (that is , martial art) . i 'm sure the guy 's good , and he seems like a nice guy on talk shows , although a tad haughty , but these movies he makes are all the same : a guy who is basically indestructable , is maybe wounded supposedly mortally , then comes back with a vengeance and goes buddha on all the baddies asses (although i kinda liked " under siege ") . of course , this one , as a change , has a " message " that is drilled into our mind . .. of course , after he blows up a lot of stuff and kills a bunch of people . so why do i watch his crap ? i usually do n't . i will never , and you can hold me to this , i will never pay to see this man 's movies , unless , and only unless , he 's in a supporting role (i. e. " executive decision ") and i 'd definitely pay if he dies (i. e. " executive decision ") . but this one has a special place in my heart . this does n't mean it 's good or that i even liked it . this was the last movie i watched with my deceased uncle , and we had one hell of a time ripping it apart a la " mystery science theatre 3000 , " and this was a couple years before i had heard of " mystery science theatre 3000 . " in this one , seagal plays a worker for a mining factory set in alaska and run by the greased - up typical shallow villain , this time played by an oscar - winner to give the movie some more clout - michael caine . it seems that caine wants to do something with his oil factory that includes him dumping oil all over inuit land . around the 20 - 30 minute point , seagal speaks up to him in what seems to be the typical speech to all the vain entrepeneurs (what with his new " fire down below , " another " message film ") , and caine has him bumped off . .. or does he ? seagal is rescued by some inuits , and falls in love with one of them , played by joan chen , who can act , hypothetically , but , for some reason , not here . one of caine 's cliched henchmen (played here with a lot of overacting by john c. mcginley) shoots the cheif of the inuit clan , and chen and seagal go on a voyage to take down the oil factory . .. literally , of course . at one point , seagal gives a wonderfully hysterical speech about how he does n't have any options but blow stuff up . he even goes as far as to say , " i do n't want to kill someone , " and in the same breath , he asks some guy where the arsenal is . i have no problem with violence . i 'm a huge john woo fan , but he paints his films with suspense , skill , style , depth , characterization , and just plain cool violence . in the films of seagal , the suspense mainly consists of the baddie attacking him stupidly , and him either wounding or killing them . at some points , they use the cliche of the talking villain , where the villain has the advantage , can shoot seagal , but begins talking by either telling him his big secret plan , or saying a corny line , to which seagal says something hokey back , and has had enough time to devise of a way to do away with them , and does . this would be okay if there were any suspense or if it did n't take itself seriously at all , like in the case of this summer 's " con air . " but seagal is serious about his skill , and of course , his message . i would n't mind if this was a message film in the way that they present it to you with evidence . but seagal has no idea how to present a film where the message is subtle , not pounded into the viewer 's mind . the villain is totally shallow and cartoonish , thus we ca n't take him and his motives seriously , and while seagal talks about being kind to the environment , he also goes ahead and blows up a square mile of rig , and kills some workers who were just doing his job . then at the end , he spends a good 10 minutes giving a speech , just in case you did n't get the message from the trailers . what seagal does n't realize is that no one takes his films seriously (although maybe a couple do) and any message he has is no only redundant , but does n't comfortably fit in his film , which is filled to the brim with hokey violence , crap suspense , stupid melodrama , and characters who have about as much emotional depth as a petri dish . as far as seagal and his acting , he 's rather boring . he squints , he kills . period . nothing else . oh , yeah , there 's corny one - liners (" i 'm gon na reach out and touch someone ! ") . of course , he 's the star , and we 're supposed to root for him and all , so he makes all the villains unbelievably stupid and a bunch of jerks . michael caine , who 's a great actor , is just supposed to yell and look cold . he does it well , i guess , but this is no " alfie . " of coure , no one was expecting that caliber of performance from him . his big henchman , john c. mcginley is kinda boring as well , but is not horrible . and we even get a small performance from that god of drill sergeants on celluloid , r. lee ermey (from " full metal jacket ") as a hired assasin squad leader who gets to say the obligatory speech about how dangerous seagal is , just for the movie trailers and for seagal 's ego . and also , look for billy bob thornton as one of ermey 's assasins . anyway , to conclude this all , to judge one of seagal 's movies is to judge all of them (except for " under siege " and " executive decision , " though the latter is not really a " seagal movie ") . they all have this same formula , they all have the same action , same villain , same plot , but this one has that message , which makes it more excrucitating to watch . i mean , if you do rent it , and i do n't reccomend you do , make sure you just skip the last 10 minutes . but i have to put it to seagal for creating a film so bad , that the last film i viewed with my uncle was a pleasurable one . my (extra star for the fun it is to watch and mock)

Review 2

like the great musical pieces of mozart himself , amadeus is a true work of art . it is one of those few movies of the 80 's that will be known for its class , its style , and its intelligence . why is this such a good film ? there are almost too many reasons to explain . the story : court composer salieri (f. murray abraham) feels waves of different emotions going through his head as wolfgang amadeus mozart (tom hulce) comes into his life as the young genius composer . salieri feels envy , and jealousy , but at the same time is fascinated with mozart 's brilliance and ingenious . we travel through mozart 's life as a composer , through his struggles , his triumphs , and ultimately , his demise . the acting : abraham is magnificient as salieri ; his acting range enables him to focus on each individual emotion and express it through his speech and body language . this performance earned him a well deserved oscar . tom hulce is interesting as well as mozart , a quirky , annoying bratty kid with an annoying laugh . he 's strong , but weak at the same time , and must be aided by his wife (elizabeth berridge) , who is good in her role , but lacks dramatic depth . jeffrey jones , in a smaller , more dignified role than such roles in stay tuned and mom and dad save the world , is cast perfectly because of his noble charm . the movie : every element of this movie works . the costumes and makeup are very memorable , as well as its stunning art direction , and unforgettable scores (adapted from mozart 's original music) . while wolfgang amadeus mozart was a genius at music , milos forman proves with his film that he is a genius of filmmaking . this movie is a classic that will be remembered for years to come . ad2am " i almost lost my nose . .. and i like it . i like breathing through it . " -jack nicholson , chinatown

Review 3

synopsis : in phantom menace the galaxy is divided into power groups whose interests will inevitably collide in later sequels . there is an overarching galactic united nations - type organization called the senate presided by a weak chancellor . within the senate two camps are at odds : a bickering , isolationist alliance called the republic and their aggressive rival the trade federation . preserving law and order are a council of jedi knights who are meanwhile searching for a prophesied chosen one of virgin birth . manipulating events behind the scenes is a dangerous , reemerging clan called the dark lords of sith , so shadowy and secretive that they comprise a " phantom " menace . jedi knight qui - gon jinn (liam neeson) and his apprentice obi - wan kenobi (ewan mcgregor) witness an invasion of teenage queen amidala 's home planet naboo and befriend a gungan named jar jar (ahmed best) . on the desert planet of tatooine the two jedi , jar jar , and amidala (natalie portman) attend a lengthy drag race involving the young boy anakin skywalker (jake lloyd) . the five protagonists try to solicit help for freeing naboo by visiting the city planet of coruscant where a lot of debate and political maneuvering takes place . can they free amidala 's helpless planet ? opinion : on tv last night i watched young , wannabe celebs pay $ 400 a ticket and come running out of theaters to bask in front of news cameras , gushing with testimonials of the phantom menace 's greatness in exchange for a few seconds of being on national television . given this kind of media mania i wondered if phantom menace , the most anticipated movie of 1999 , could possibly live up to the extraordinary hype that preceded it . does phantom menace match the exaggerated hype ? director george lucas answers , " it 's only a movie . " to me , any movie with russian - sounding accents for bad guys , jamaican accents for good guys , and middle eastern - sounding accents for seedy gamblers accents can be expected to be more tongue in cheek than profound . visually , star wars : episode i –the phantom menace (1999) is a kid show where parents can take their young ones to marvel at child - friendly cgi characters and wondrous backdrops even if the character dialogue (mostly geopolitics) is beyond the level of children . it is left to parents to patiently explain the conversation : droid origins , family lineage , the definitions of terms like blockade , appeasement , federation , alliance , symbiosis , satellite - controlled robots et cetera . at least this much is clear : there 's plenty of eye candy , and in the last few minutes it 's good guys and joe camel lookalikes versus a caped , horned red devil character and his mechanical hordes . weaknesses : weaknesses lie in the writing and in the performance . at first it seems like the film is to be an invasion story , but then phantom takes an hour - long detour to cover one chariot race before returning to the invasion theme . this dilutes the central story . additionally , smaller scenes seem written self consciously , as if they were added more to fill us in on extraneous background information for other movies rather than form an integral part of the present movie . veteran actors liam neeson and ewan mcgregor noticeably outperform the other acting leads . better ensemble chemistry between the five leads and background information that is central to a tight story line could have made have given phantom stronger performances and storytelling punch . strengths : on the bright side phantom menace as a big - budget production is far ahead of the competition in terms of making whimsical creatures , worlds and vehicles appear real . the film boasts sophisticated , top - of - the - line visuals and quality exotic costumes , a musical score entertaining enough to stand alone , and three worthwhile sequences in the second half . bottom line ? seeing the film is entertaining and informative , like a visual theme park with star wars filler information serving as dialogue between impressive money shots . we are bound to be completely inundated by star wars publicity , music and tie - ins for the next few months .

Selbstständigkeitserklärung

Hiermit versichere ich, Tobias Christian Nauen, dass ich diese Arbeit selbstständig verfasst habe und keine weiteren als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle Stellen der Arbeit, die wörtlich oder sinngemäß aus anderen Quellen übernommen wurden, habe ich als solche kenntlich gemacht. Zudem wurde die Arbeit bisher in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegt.

_____ _____

Ort, Datum Unterschrift