Explainable Automated Graph Representation Learning with Hyperparameter Importance

Xin Wang [1] Shuyi Fan [1] Kun Kuang [2] Wenwu Zhu [1]

Abstract

Current graph representation (GR) algorithms require huge demand of human experts in hyperparameter tuning, which significantly limits their practical applications, leading to an urge for automated graph representation without human intervention. Although automated machine learning (AutoML) serves as a good candidate for automatic hyperparameter tuning, little literature has been reported on automated graph presentation learning and the only existing work employs a black-box strategy, lacking insights into explaining the relative importance of different hyperparameters. To address this issue, we study explainable automated graph representation with hyperparameter importance in this paper. We propose an explainable AutoML approach for graph representation (e-AutoGR) which utilizes explainable graph features during performance estimation and learns decorrelated importance weights for different hyperparameters in affecting the model performance through a non-linear decorrelated weighting regression. These learned importance weights can in turn help to provide more insights in hyperparameter search procedure. We theoretically prove the soundness of the decorrelated weighting algorithm. Extensive experiments on real-world datasets demonstrate the superiority of our proposed e-AutoGR model against state-of-the-art methods in terms of both model performance and hyperparameter importance explainability.

[1]Department of Computer Science and Technology, Tsinghua University, Beijing, China [2]College of Computer Science and Technology, Zhejiang University, Hangzhou, China. Correspondence to: Xin Wang <xin_wang@tsinghua.edu.cn>, Wenwu Zhu <wwzhu@tsinghua.edu.cn>.

Proceedings of the 38th International Conference on Machine Learning, PMLR 139, 2021. Copyright 2021 by the author(s).

1. Introduction

Graph representation (GR) has attracted lots of research attentions from both academia and industry, due to its wide applications in the past decade (Tenenbaum et al., 2000; Hamilton et al., 2017b; Zhu et al., 2020). The increasing amount of graph data in recent years has posed great challenges for graph representation algorithms which normally contain a large number of hyperparameters to be optimized through careful tuning. Automated machine learning (AutoML) serves as a good candidate to reduce the human effort necessary for model tuning through automating the process of hyperparameter optimization and improving the performance of machine learning models (Hutter et al., 2019). Existing AutoML models normally resort to a group of performance evaluation functions which take hyperparameters as input to estimate the potential performance given various configurations of hyperparameters. These evaluation functions are optimized in a black-box manner and unable to explain the importance of each hyperparameter in affecting the model performance. The only work applying AutoML to graph representation (Tu et al., 2019) adopts Gaussian Process with a predefined kernel as the covariance to estimate the hyperparameter performance, failing to explain how each hyperparameter contributes to the estimated performance and why a certain value of each hyperparameter is chosen for the next round of evaluation.

However, the complex relational information carried in large-scale graphs leads to an urge for questions such as *what is the importance of each hyperparameter, how it contributes to the model performance and why one particular value is selected for a hypeparameter in the next trial*, which serve as the key to scientific discoveries in automated graph representation learning.

To answer these questions, we investigate explainable automated graph representation with hyperparameter importance in this paper. Given that graphs in real world usually contain billions of nodes and edges and the computational costs of running graph representation algorithms on these graphs are proportional to the number of nodes and edges, it will not be realistic to search and optimize hyperparameters through directly testing a graph representation algorithm on the original large-scale graph many times. Therefore, we propose an explainable automated graph representation (e-AutoGR) framework, which first runs the target graph representation algorithm on several sampled subgraphs of much smaller sizes for some trials, and then utilizes the running results obtained from these subgraphs to infer the optimal hyperparameters for the original graph. We adopt six explainable graph properties as explicit features to calculate the similarities between the sampled subgraphs and the original graph such that e-AutoGR is able to automatically determine the number of trials on each subgraph. These explainable graph features together with hyperparameters are passed into a non-linear evaluation function to calculate the estimated model performance under the current hyperparameter configuration. In order to explain hyperparameter importance in affecting the model performance, we propose a hyperparameter decorrelation algorithm (HyperDeco) to decorrelate the mutual influences among different hyperparameters on the estimated model performance, given graph features and some hidden unobserved latent factors. We would like to point out that there exists one work on non-graph problem in literature (Hutter et al., 2014) aiming to assess hyperparameter importance through providing a classic functional ANOVA technique for Bayesian Optimization with random forest prediction, which limits its applicability in other predicting approaches. In contrast, our proposed e-AutoGR model does not have this limitation and benefits in its flexible applicability to any predicting function for relational graph data.

As such, our proposed e-AutoGR framework is able to explain how important each hyperparameter is to the model performance by learning decorrelated importance weights through a non-linear hyperparameter decorrelated weighting regression. These learned decorrelated importance weights will in turn help to increase the explainability in hyperparameter search procedure. We further present a theorem as well as the relevant proof to guarantee the theoretical soundness of the decorrelated weighting algorithm. Compared with existing methods, the proposed e-AutoGR framework benefits in more explainable hyperparameter search and performance evaluation strategy with insights on hyperparameter importance.

To summarize, this paper makes the following contributions:

- We investigate the problem of explainable automated machine learning (AutoML) for graph representation with hyperparameter importance, to the best of our knowledge, for the first time.

- We propose an explainable automated hyperparameter optimization framework (e-AutoGR) which is capable of explaining how each hyperparameter contributes to the model performance as well as why some certain values of hyperparameters are chosen for the next round of evaluation, through a novel non-linear hyperparameter decorrelated weighting regression. The proposed explainable e-AutoGR framework also employs fully explainable graph features to dynamically determine the number of trials on each sampled subgraphs.

- We theoretically validate the correctness of the proposed hyperparameter decorrelation algorithm.

- We conduct extensive experiments on several real-world datasets to demonstrate the superiority of e-AutoGR over various state-of-the-art methods in terms of both model explainability and performance.

2. Related Work

In this section, we will discuss related works on graph representation and automated machine learning (AutoML).

Graph Representation. Graph representation (Roweis & Saul, 2000; Tenenbaum et al., 2000; Hamilton et al., 2017b) aims at learning low-dimension vector representations of nodes to facilitate a better understanding for semantic relationships among nodes in graphs. Sampling based methods (Grover & Leskovec, 2016; Perozzi et al., 2014) employ a truncated random walk to generate node sequences. These node sequences can be regarded as sentences and fed into the skip-gram model to learn low-dimension node representations. In addition, topological graph structure can also be taken into account by factorization based models, through requiring the node representations to be able to preserve various topological properties in the original graph, which includes but not limits to different order proximities (Zhang et al., 2018) and asymmetric transitivity (Ou et al., 2016) etc. With the success of deep neural networks, deep autoencoder based models are proposed to preserve the highly non-linear first order and second order proximities (Tang et al., 2015; Wang et al., 2016; Cao et al., 2015), capture uncertainties (Zhu et al., 2018) or handle triplet relations (Tu et al., 2018). Recently, inspired by Graph Convolutional Network (GCN) (Kipf & Welling, 2016) which combines convolution operators with graph to learn node representations in an end-to-end manner, a wide variety of graph neural network based models have been proposed to take both topological structures and node attributes into consideration (Hamilton et al., 2017a; Velickovic et al., 2017; You et al., 2018; Ma et al., 2019). The massive properties that need to be preserved naturally require a large number of hyperparameters to be controlled manually in graph representation algorithms, posing a high demand for automating the process of hyperparameter optimization in graph representation. We would like to point out that the literature on mining graphs or networks has two names: graph representation and network embedding. We remark that graph and network all refer to the same kind of structure, although each of them may have its own terminology. Therefore we do not distinguish between them here.

Automated Machine Learning. Automated machine learning (AutoML) (Feurer et al., 2015; Yao et al., 2018; Hutter et al., 2019) aims to set manual efforts free from tuning machine learning models, ranging from data preprocessing (Fang et al., 2017), feature engineering (Kanter & Veeramachaneni, 2015), model selection (Escalante et al., 2009; 2010), hyperparameter optimization (Thornton et al., 2013; Mantovani et al., 2016; Luo, 2016; Sanders & Giraud-Carrier, 2017; Golovin et al., 2017) and neural architecture search (Zoph & Le, 2017; Liu et al., 2019; Guo et al., 2019; Wu et al., 2019; Cai et al., 2018; Real et al., 2019; Liu et al., 2018). Neural architecture essentially can be regarded as a special type of hyperparameter which is indifferentiable. Among all these tasks, hyperparameter optimization is the most relevant to our proposed framework in this work. There are several groups of methods widely used in hyperparameter optimization, among which grid search and random search (Bergstra & Bengio, 2012) are two most straightforward ways to search a good configuration of hyperparameters from a set of candidates without taking the history results into account. Bayesian optimization (Snoek et al., 2012), as one of the most popular sequential model-based optimization (SMBO) (Hutter et al., 2011) approaches capable of utilizing history results, adopts Gaussian process to model the surrogate function that approximates the relations between hyperparameters and their expected performance. However, all these methods are optimized in a black-box manner. The only work on AutoML for graph representation (Tu et al., 2019) uses Gaussian Process to estimate the hyperparameter performance, and can hardly explain how each hyperparameter contributes to the model performance and why one particular value is chosen for a hyperparameter to perform the next trial of evaluation.

3. Explainable Automated Graph Representation: e-AutoGR

We first give a brief overview of the problem definition, followed by the two core components of e-AutoGR: i) subgraph sampling and computation allocation and ii) weighting regression with hyperparameter decorrelation.

3.1. Problem Definition

Given a graph $G = (V, E)$ whose nodes and edges are denoted by V and E respectively. Assuming we sample s subgraphs from the original graph G, we denote $G_i = (V_i, E_i)$ where $i = 1, 2, 3, \ldots, s$ as the i-th subgraph. Let $\mathsf{P}_R(\Lambda, G)$ denotes the performance of graph representation algorithm R with hyperparameter configuration Λ on graph G, where the performance can be accuracy of node classification, link prediction and graph reconstruction etc. We adopt the accuracy of node classification and link prediction, two most popular applications for graph representation, as the performance to be estimated in this paper. As we point out earlier, large-scale graphs in real world tend to have quite large $|V|$ and $|E|$, resulting in very high computational costs of running graph representation algorithms on these graphs. Thus it will be unrealistic to search and optimize hyperparameters through directly evaluating the performance of a graph representation algorithm on the original graph for many times. As such, e-AutoGR first samples several subgraphs which is much smaller in size compared with the original, then runs the selected graph representation algorithm with different hyperparameter configurations on each of subgraph for different times depending on how similar the sampled subgraph is to the original graph, finally utilizes the results of performance obtained from these subgraphs to infer the optimal hyperparameter configuration for the original graph.

Given a graph representation algorithm R and a target graph G, the goal of e-AutoGR is to find out a hyperparameter configuration Λ^* having the optimal performance $\mathsf{P}_R(\Lambda^*, G)$ for algorithm R on graph G based on the results from running algorithm R on each of the sampled subgraphs G_i several times. We additionally require e-AutoGR to be capable of explaining how each hyperparameter contributes to the model performance as well as why one particular value is chosen for a hyperparameter to perform the next trial of evaluation.

3.2. Subgraph Sampling and Computation Allocation

We start by sampling subgraphs from the original graph. Although the smaller sizes of subgraphs enable us to run algorithm R many more times on subgraphs, it is not desired to abuse the computational resources. Thus we set a budget n for the maximum total number of times to run R on subgraphs and dynamically decide how many times R can be executed on each subgraph within the budget.

3.2.1. Subgraph Sampling

Each subgraph is sampled through executing several random walks on the original graph and different subgraphs can therefore be obtained by varying the the length of each random walk. For subgraph $G_i = (V_i, E_i)$, V_i is the set of nodes traversed by the several corresponding random walks and $E_i = \{(u, v)|u \in V_i \wedge v \in V_i \wedge (u, v) \in E\}$. To ensure the sampled subgraphs can cover the property and diversity of the original graph as much as possible, the random walks for sampling different subgraph are forced to start from different regions of the original graph. In practice, the random walks for different subgraphs can start from nodes with different labels when the node labels are available and start from different communities detected through some community detection methods (Fortunato, 2010) when the node labels are unavailable. Without loss of generality, let's assume e-AutoGR samples s subgraphs in total.

3.2.2. Computation Allocation on Subgraphs

Given the budge n and s sampled subgraph $G_i, i = 1, 2, 3, ..., s$, we for each subgraph G_i dynamically decide a personalized number of times t_i to execute R such that $n = \sum_{i=1}^{s} t_i$. For the sake of explainability, we adopt six fully explainable graph feature (Bonner et al., 2016) to conduct this allocation task:

- Number of nodes: $|V|$
- Number of edges: $|E|$
- Number of triangles: The number of nodes forming a triangle which is a set of three nodes connected with each other by three edges.
- Global clustering coefficient: $\frac{3*Number\ of\ triangles}{Number\ of\ triplets}$, where a triplet is three nodes that are connected by either two or three (forming a triangle) undirected edges.
- Maximum total degree value: The total number of edges the most connected node in the graph has to other nodes.
- Number of Components: The total number of components contained in the graph, where a component is a subgraph in which there is a possible path between every node, while nodes from different components have no path connecting them.

Theses six explainable graph features thus form a 1×6 dimension vector and let us denote the feature vector of original graph G and subgraph G_i as $\mathbf{f}$ and $\mathbf{f}^i$ respectively. We then adopt the *Canberra Distance* to calculate the similarity between each subgraph G_i and the original graph G:

$$g^i = d(\mathbf{f}^i, \mathbf{f}) = \sum_{k=1}^{6} \frac{|\mathbf{f}_k^i - \mathbf{f}_k|}{|\mathbf{f}_k^i| + |\mathbf{f}_k|}. \quad (1)$$

The number of times executing algorithm R on subgraph G_i is then determined as follows:

$$t_i = \left\lfloor n * \frac{g^i}{\sum_{k=1}^{s} g^k} \right\rfloor. \quad (2)$$

Let $r = n - \sum_{k=1}^{s} t_k$, where r denotes the remaining number of times we can execute algorithm R. We then select the first r subgraphs having the largest g^i, and let $t_i = t_i + 1$ for each G_i in the selected r subgraphs.

We remark that other distance metrics include Bray, Correlation, Chebyshev, Cosine and Manhattan etc., but they are insensitive when feature vectors are highly similar, or produce unintuitive results such as a high similarity score for highly dissimilar graphs. The Canberra distance can detect changes close to zero, making it ideal for detecting small variations (e.g., similarity of subgraphs close to each other) in graph topology. Our assumption is that hyperparameters for more similar subgraph will be more transferrable to original graph.

We would also like to point out that increasing the size of subgraph will not always result in getting more computation resources. The reason is as follows. We aim to sample a series of representative subgraphs that share "some" similar properties with the original large-scale graph. We sample each subgraph based on several random walks. And we vary the length of each random walk to obtain subgraphs of various sizes. To ensure different subgraphs diversely preserve different properties of the original graph, the starting points of random walks are explicitly chosen to be at the different regions of the original graph. As different regions (e.g., local communities or local topologies with same category/label) may be of different sizes, larger subgraphs will not always be more similar to the original graph.

3.3. Weighting Regression with Hyperparameter Decorrelation

After running graph representation algorithm R on sampled subgraphs for different times, we obtain n samples with different configurations of hyperparameters and graph features as well as their model performance. If algorithm R runs on subgraph G_i for t_i times, where $i = 1, 2, 3, \ldots, s$, then $n = \sum_{i=1}^{s} t_i$. To infer the optimal hyperparameter configuration for the original graph, we will need to learn a non-linear mapping from hyperparameters to performance with the importance of each hyperparameter in influencing the performance. Given that the n samples are obtained from different subgraphs rather than the original graph, the non-linear mapping will also need to take graph features as input to guide the learning process. Therefore, the performance mathematically depends on hyperparameters, graph features and some other unobserved hidden factors as well as their mutual correlations simultaneously, which makes it very difficult to calculate and learn the real importance of each hyperparameter in influencing the performance.

To tackle this difficulty, we propose a hyperparameter decorrelation weighting regression (HyperDeco) algorithm to decorrelate the mutual influences among different hyperparameters and graph features as well as some hidden unobserved latent factors on the estimated model performance. Upon decorrelation, it will be possible to learn each hyperparameter's true importance in impacting the performance.

3.3.1. Hyperparameter Decorrelation

Assuming we have n samples, each of which is represented as a $1 \times p$ vector where $p = p_1 + p_2$. The first p_1 dimensions contain p_1 hyperparameters and the last p_2 dimensions contain graph features obtained from Section 3.2.1 ($p_2 = 6$ in our case). Let $\mathbf{A} \in \mathbb{R}^{n\times p_1}$ denote the matrix of hyperparameters, where n refers to the sample size and p_1 means the dimension of hyperparameters. Let $\mathbf{B} \in \mathbb{R}^{n\times p_2}$ denote the graph features with dimension p_2. For simplify, let $\mathbf{X} = [\mathbf{A}, \mathbf{B}] \in \mathbb{R}^{n\times p}$.

In this paper, we focus on hyperparameters' first moment (i.e., mean) and propose to decorrelate all the predictors by sampling reweighting in the training environment. The reason of focusing on the first moment is as follows, Correlation can be defined as $Correlation = Cov[X, Y]/d[X]d[Y]$, where $Cov[X, Y] = E[XY] - E[X]E[Y], E, d$ are expectation (mean and first moment) and standard deviation respectively. $E[XY] = E[X]E[Y]$ leads to $Correlation = 0$ (i.e.,decorrelation), indicating the sufficiency of first moment. Specifically, we propose a *hyperparameter decorrelation* regularizer for learning the sample weight γ as follows:

$$\min_{\gamma} \sum_{j=1}^{p_1} \left\| \mathbb{E}[\mathbf{A}_{,j}^T \mathbf{\Sigma}_\gamma \mathbf{X}_{,-j}] - \mathbb{E}[\mathbf{A}_{,j}^T \gamma]\mathbb{E}[\mathbf{X}_{,-j}^T \gamma] \right\|_2^2, \tag{3}$$

where $\gamma \in \mathbb{R}^{n\times 1}$ are sample weights satisfying $\sum_{i=1}^n \gamma_i = n$, $\Sigma_\gamma = diag(\gamma_i, \ldots, \gamma_n)$ is the corresponding diagonal matrix, $\mathbf{A}_{,j}$ denotes the j^{th} column/hyperparameter in $\mathbf{A}$, and $\mathbf{X}_{,-j} = \mathbf{X} \setminus \mathbf{A}_{,j}$ means all the remaining variables by removing the hyperparameter $\mathbf{A}_{,j}$ in $\mathbf{X}$.[1] The summand represents the loss due to correlation between hyperparameter $\mathbf{A}_{,j}$ and all other hyperparameters together with graph features $\mathbf{X}_{,-j}$. Note that, only first moment (i.e., mean) is considered in Eq (3), but it is sufficient for hyperparameter decorrelation. And higher-order moments can be easily incorporated.

The following theoretical results show that our hyperparameter decorrelation regularizer can make the variables in $\mathbf{A}$ become uncorrelated with variables in $\mathbf{X}$ by sample reweighting, hence reduce the correlation among covariates in the training environment, improving the accuracy and explainability on importance estimation for hyperparameter.

With $\sum_{i=1}^n \gamma_i = n$, we can denote the loss in Eq (3) as:

$$\mathcal{L}_{Deco} = \textstyle\sum_{j=1}^{p_1} \left\| \mathbf{A}_{,j}^T \mathbf{\Sigma}_\gamma \mathbf{X}_{,-j}/n - \mathbf{A}_{,j}^T \gamma/n \cdot \mathbf{X}_{,-j}^T \gamma/n \right\|_2^2. \tag{4}$$

Lemma 1 *If the number of covariates p_1 and p_2 is fixed, then there exists a sample weight $\gamma \succeq 0$ such that*

$$\lim_{n\to\infty} \mathcal{L}_{Deco} = 0 \tag{5}$$

with probability 1. *In particular, a solution γ to Eq (5) is $\gamma_i^\star = \frac{\Pi_{j=1}^p \hat{f}(\mathbf{X}_{i,j})}{\hat{f}(\mathbf{X}_{i,1},\cdots,\mathbf{X}_{i,p})}$, where $\hat{f}(x_{\cdot,j})$ and $\hat{f}(x_{\cdot,1},\cdots,x_{\cdot,p})$ are the Kernel density estimators.*[2]

[1] We obtain $\mathbf{X}_{,-j}$ in experiment by setting the value of $\mathbf{A}_{,j}$ in $\mathbf{X}$ as $zero$.

[2] In detail, $\hat{f}(x_{i,j}) = \frac{1}{nh_j}\sum_{i=1}^n k\left(\frac{\mathbf{X}_{i,j}-x_{i,j}}{h_j}\right)$, where $k(u)$ is a kernel function and h_j is the bandwidth parameter for covariate $\mathbf{X}_j$; and $\hat{f}(x_i) = \frac{1}{n|H|}\sum_{i=1}^n K\left(H^{-1}(\mathbf{X}_i - x_i)\right)$, where $K(u)$ is a multivariate kernel function, $H = diag(h_1, \cdots, h_p)$ and $|H| = h_1 \cdots h_p$.

Proof 1 *See supplementary file.*

But the solution γ that satisfies Eq (5) in Lemma 1 is not unique. To address this problem, we propose to simultaneously minimize the variance of γ and restrict the sum of γ in our regularizer as follows:

$$\hat{\gamma} = \arg\min_{\gamma\in\mathcal{C}} \mathcal{L}_{Deco} + \frac{\lambda_3}{n}\textstyle\sum_{i=1}^n \gamma_i^2 + \lambda_4(\frac{1}{n}\sum_{i=1}^n \gamma_i - 1)^2, \tag{6}$$

where $\mathcal{C} = \{\gamma : |\gamma_i| \le c\}$ for some constant c.

Then, we have following theorem on our hyperparameter decorrelation regularizer in Eq (6).

Theorem 1 *The solution $\hat{\gamma}$ defined in Eq (6) is unique if $\lambda_3 n \gg p^2 + \lambda_4$, $p^2 \gg \max(\lambda_3, \lambda_4)$ and $|\mathbf{X}_{i,j}| \le c$ for some constant c.*

Proof 2 *See supplementary file.*

With Lemma 1 and Theorem 1, we can derive the following property of the $\hat{\gamma}$ given by Eq (6).

Property 1. *When p_1 and p_2 is fixed, $n \to \infty$, $\lambda_3 n \gg p^2 + \lambda_4$, and $p^2 \gg \max(\lambda_3, \lambda_4)$, the variables in $\mathbf{A}$ become uncorrelated with the variables in $\mathbf{X}$ by sample reweighting with $\hat{\gamma}$.*

3.3.2. Decorrelated Weighting Regression

Upon obtaining $\hat{\gamma}$ from hyperparameter decorrelation regularizer in Eq (6), the learning of non-linear mapping from hyperparameters and graph features to performance can be formulated as a weighted non-linear regression problem:

$$\hat{\Theta} = \arg\min_{\Theta} \sum_{i=1}^n \hat{\gamma}_i \cdot (Y_i - \phi(\mathbf{X}_{i,}; \Theta))^2, \tag{7}$$

where Y_i is the observed performance for $X_{i,}$ and $\phi(; \Theta)$ is the non-linear mapping formulated through a multi-layer perceptron (MLP) with parameter Θ. By Property 1, the summation of the absolute values of each hyperparameter's weights towards neurons in the hidden layer of MLP can be approximately adopted as the unbiased importance of the hyperparameter in affecting the performance, since $\hat{\gamma}$ decorrelates variables between $\mathbf{A}$ and $\mathbf{X}$.

By combining the objective functions of the hyperparameter decorrelation regularizer in Eq (6) and the weighted non-linear regression in Eq (7), we propose a hyperparameter decorrelation weighting regression (HyperDeco) algorithm

Algorithm 1 Hyperparameter Decorrelation Weighting Regression (HyperDeco)

```
1:  Input: Observed $\mathbf{X} = [\mathbf{A}, \mathbf{B}]$ and performance $Y$, where $\mathbf{A}$ denotes hyperparameters and $\mathbf{B}$ denotes graph features.
2:  Output: Updated parameters $\gamma$, $\Theta$.
3:  Initialize parameters $\gamma^{(0)}$ and $\Theta^{(0)}$,
4:  Calculate loss function with parameters $(\gamma^{(0)}, \Theta^{(0)})$,
5:  Initialize the iteration variable $t \leftarrow 0$,
6:  repeat
7:      $t \leftarrow t + 1$,
8:      Update $\gamma^{(t)}$ with gradient descent by fixing $\Theta$,
9:      Update $\Theta^{(t)}$ with gradient descent by fixing $\gamma$,
10:     Calculate loss function with parameters $(\gamma^{(t)}, \Theta^{(t)})$,
11: until Loss function converges or max iteration is reached
```

to jointly optimize γ and Θ as follows:

$$\begin{aligned} & \min_{\gamma,\Theta} \sum_{i=1}^{n} \gamma_i \cdot (Y_i - \phi(\mathbf{X}_{i,}; \Theta))^2 \qquad (8) \\ s.t \quad & \sum_{j=1}^{p_1} \left\| \mathbf{A}_{,j}^T \mathbf{\Sigma}_\gamma \mathbf{X}_{,-j}/n - \mathbf{A}_{,j}^T \gamma/n \cdot \mathbf{X}_{,-j}^T \gamma/n \right\|_2^2 < \lambda_2 \\ & |\Theta|_1 < \lambda_1, \; \frac{1}{n}\sum_{i=1}^{n} \gamma_i^2 < \lambda_3, \\ & (\frac{1}{n}\sum_{i=1}^{n} \gamma_i - 1)^2 < \lambda_4, \; \gamma \succeq 0, \end{aligned}$$

where n denotes the sample size, p_1 refers to the dimension of hyperparameters $\mathbf{A}$. $\mathbf{A}_{i,}$ and $\mathbf{A}_{,j}$ represent the i^{th} sample and the j^{th} variable in $\mathbf{A}$, respectively. The term $\gamma \succeq 0$ constrains each sample weight to be non-negative. With term $\frac{1}{n}\sum_{i=1}^{n} \gamma_i^2 < \lambda_3$, we reduce the variation of the sample weights. The term $(\frac{1}{n}\sum_{i=1}^{n} \gamma_i - 1)^2 < \lambda_4$ avoids all sample weights to be $zero$.

3.3.3. Optimization

To optimize our HyperDeco algorithm in Eq (8), we initialize sample weights $\gamma_i = 1$ for each sample i and regression coefficient $\Theta = \mathbf{0}$. Once the initial values are given, in each iteration, we first update γ by fixing Θ, then update Θ by fixing γ until the objective function in Eq (8) converges. The whole algorithm is summarized in Algorithm 1.

3.4. The e-AutoGR Framework: Optimizing Hyperparameters with Importance Explanation

We resort to the MLP optimized through HyperDeco (Algorithm 1) to predict the hyperparameters that may lead to the optimal performance on the original graph G whose graph features can be easily obtained according to Section 3.2.2. We follow three steps:

- **Step 1:** We run HyperDeco with the observed samples (i.e., $\mathbf{X} = [\mathbf{A}, \mathbf{B}]$) to get the updated γ and Θ.

Algorithm 2 Explainable Automated Graph Representation (e-AutoGR)

```
1:  Input: Graph $G$, Graph representation algorithm $R$.
2:  Output: The optimal hyperparameter configuration $\Lambda^*$.
3:  Sample $s$ subgraphs $G_i, i = 1, 2, \ldots, s$ from original graph $G$ according to Section 3.2.1.
4:  Decide $t_i$ for each $G_i$ according to Section 3.2.2.
5:  Execute algorithm $R$ on each subgraph $G_i$ for $t_i$ times and obtain hyperparameter matrix $A$ and graph feature matrix $B$ as well as the performance vector $Y$.
6:  Initialize $Count = T$.
7:  repeat
8:      Execute Step 1 to Step 3 in Section 3.4.
9:      $Count = Count - 1$.
10: until $Count == 0$
```

- **Step 2:** We first rank hyperparameters in a descending order by their importance in affecting the performance, then optimize the most important hyperparameter with the others fixed. We uniformly sample 1000 possible values for the hyperparameter to be optimized, and search for the one with the best predicted performance through the learned non-linear mapping by MLP. Thus we traverse every hyperparameter one by one according to their importance and each time optimize one hyperparameter by fixing the others. The idea of using weights for the first layer of MLP to estimate the variable importance is inspired by previous work (Hassanpour & Greiner, 2019), and we conduct normalization for each hyperparameter i.e., $(x - min)/(max - min)$,to deal with the scale issue.

- **Step 3:** We execute algorithm R with the predicted hyperparameter configuration on the original graph G and obtain the true performance. The new hyperparameter configuration, original graph features as well as their performance can in turn be appended to the existing observed samples as a new observed sample.

We repeat **Step 1** to **Step 3** T times and adopt the hyperparameter configuration having the best performance so far as the current values for hyperparamters. Algorithm 2 summarizes the proposed e-AutoGR framework.

Complexity Analysis. The time complexity of HyperDeco is $O(cnp^2)$, where c is the number of iterations, n is the sample size and p is the sample dimension. This is because the complexity of calculating Eq (8) is $O(np^2)$, and the cost of updating γ and Θ are $O(np^2)$ and $O(np)$ respectively. The total time complexity of e-AutoGR is $O(T|E|)$, which is dominated by executing graph representation algorithm R whose complexity is $O(|E|)$ for T times. Actually, $T \ll |E|, s \ll |E|, n \ll |E|$, $p \ll |E|$ and $|E_i| \ll |E|$, which makes the overall time complexity of sampling subgraphs, executing R on subgraphs and running HyperDeco algorithm negligible when comparing with executing R on the original graph G.

Discussions. We close this section by pointing out that the proposed e-AutoGR framework makes use of six explainable graph features for subgraph sampling and algorithm executing frequency allocation. The HyperDeco algorithm takes hyperparameters and graph features as input to learn a decorrelated non-linear mapping to predict performance with hyperparameter importance. The explainable graph features serve as a kind of transferable meta-information across different subgraphs and the original graph. As such, our proposed e-AutoGR exhibits much more explainability compared with existing literature. We also inevitably introduce some extra hyperparameters for MLP, which is the same case for existing works utilizing other techniques or tools. We argue that the MLP used by e-AutoGR only possesses a single hidden layer with 5 neurons whose hyperparameters are much easier and far less costly to optimize than those complicated graph representation algorithms.

Almost all hyperparameters in major graph representation algorithms can be supported by e-AutoGR, including model specific hyperparameters such as number/length of random walks and window size in DeepWalk (Perozzi et al., 2014), weights of different order proximity in AROPE (Zhang et al., 2018) etc. as well as some general hyperparameters such as number of training epochs, learning rate, and dropout rate for deep neural network based models. We choose the six explainable graph features based on four criteria: i) Scalability, ii) Sensitivity to graph size, iii) Sensitivity to similar topologies, iv) Label free (being able to perform comparisons without the need for labeled datasets). The selected six features can make a very good balance between topological sensitivity and runtime.

4. Experiments

Comparable Approaches. The following approaches including e-AutoGR are compared. 1) *e-AutoGR*: This is our proposed explainable automated graph representation learning framework with decorrelated hyperparameter importance in this work. 2) *AutoNE (Tu et al., 2019)*: This is the only existing work so far on AutoML for graph representation using Gaussian Process with a predefined kernel to estimate the hyperparameter performance. 3) *Bayesian optimization (BayesOpt) (Snoek et al., 2012)*: This is the sequential model-based optimization method adopting Gaussian process to model the surrogate function that approximates the relations between hyperparameters and their expected performance. 4) *Random search (Bergstra & Bengio, 2012)*: This is one of the most widely adopted strategies for hyperparameter optimization.

For e-AutoGR and AutoNE, the number subgraphs to be sampled, i.e., s, is set to 5 in our experiment, the size of each subgraph is uniformly chosen between $5\%|V|$ and $20\%|V|$, then five random walks will start in parallel to sample the subgraph and stop when the size of the subgraph reaches the chosen size. We repeat each method for five times and report the mean as results.

Graph Representation Algorithms. We adopt the same three graph representation models[3] used by AutoNE as algorithm R for test: i) DeepWalk (Perozzi et al., 2014) is the most representative sampling based graph representation algorithm. ii) AROPE (Zhang et al., 2018) is a representative factorization based graph representation algorithm. iii) GCN (Kipf & Welling, 2016) is one of the most representative deep neural network based graph representation algorithms. The number of trials executing graph representation algorithm R on original graph G, i.e., T is set to 10, which is also consistent with AutoNE.

Hyperparameter Search Space. For DeepWalk, the number of random walks is considered between 2 and 20, the length of each random walk is considered between 2 and 80 and the window size is considered between 2 and 20. For AROPE, the weights of different order proximity are considered chosen between 0.0001 and 0.1. For GCN, the number of training epochs is chosen between 2 and 300, the number of neurons in each layer is chosen between 2 and 64, the learning rate is chosen between 0.0001 and 0.1, the dropout rate is chosen between 0.1 and 0.9 and the norm-2 regularizations chosen between 0.00001 and 0.001.

Datasets. Our experiments are performed on three real-world datasets: i) BlogCatalog[4] is a social network which contains $10,312$ nodes, $333,983$ edges and 39 categories; ii) Wikipedia[5] is a word co-occurrence network with $4,777$ nodes, $184,812$ edges and 40 labels; iii) Pubmed[6] is a citation network consisting of $19,717$ nodes, $44,338$ edges, 500-dimension node features and 3 classes. As GCN requires both graph topology and node features as input, we will only test GCN on Pubmed rather than BlogCatalog and Wikipedia which do not contain node features.

4.1. Experimental Results

Node classification and link prediction, two popular tasks for graph representation are used to measure the performance of each tested graph representation algorithm. The learned representation for each node can be passed though a logistic regression to do classification and the inner product between the learned representations of two nodes can be

[3] i) https://github.com/phanein/deepwalk;ii) https://github.com/ZW-ZHANG/AROPE;iii) https://github.com/tkipf/gcn

[4] http://socialcomputing.asu.edu/datasets/BlogCatalog3

[5] https://snap.stanford.edu/node2vec/

[6] https://github.com/tkipf/gcn/tree/master/gcn/data

Table 1. Best performance for each comparable automated graph representation approach on different datasets in terms of *link prediction* and *node classification* tasks over various graph representation algorithms. Bold font denotes the best approach.

Dataset	Algorithm	Task	e-AutoGR	AutoNE	Random	Bayesian
BlogCatalog	Deepwalk	Link Prediction	**0.871817**	0.792662	0.803191	0.807158
BlogCatalog	Deepwalk	Classification	**0.414682**	0.414234	0.411551	0.407449
BlogCatalog	AROPE	Link Prediction	**0.852612**	0.851921	0.846578	0.851878
BlogCatalog	AROPE	Classification	**0.326721**	0.325060	0.326020	0.326428
Wikipedia	Deepwalk	Link Prediction	0.729228	**0.729330**	0.696462	0.713133
Wikipedia	Deepwalk	Classification	**0.519657**	0.509920	0.503319	0.502617
Wikipedia	AROPE	Link Prediction	**0.709392**	0.709383	0.703443	0.707619
Wikipedia	AROPE	Classification	0.529743	0.529011	**0.530418**	0.529732
Pubmed	Deepwalk	Link Prediction	**0.873301**	0.867633	0.853459	0.851824
Pubmed	Deepwalk	Classification	**0.810916**	0.810368	0.809199	0.810417
Pubmed	AROPE	Link Prediction	0.791435	**0.796737**	0.790228	0.790123
Pubmed	AROPE	Classification	**0.727413**	0.725412	0.725789	0.726411
Pubmed	GCN	Classification	**0.708830**	0.708712	0.708719	0.707918

used to conduct link prediction for these two nodes. We randomly withhold 20% of the node labels and 20% of the edges for testing in node classification and link prediction task, respectively. Micro-F1 score (Chinchor, 1992; Sasaki, 2007) and area under the curve (AUC) (Myerson et al., 2001) are used as performance evaluation metrics for node classification and link prediction, respectively.

We measure the performance of different hyperparameter optimization methods with various numbers of trials to execute the selected graph representation algorithm R on the original graph G. Table 1 presents the results of best performance for each comparable approach over different testing graph representation algorithms in terms of *link prediction* and *node classification* tasks on the three datasets. We observe that the proposed e-AutoGR model beats each baseline method in all but three testing cases, demonstrating its capability of achieving the best model performance.

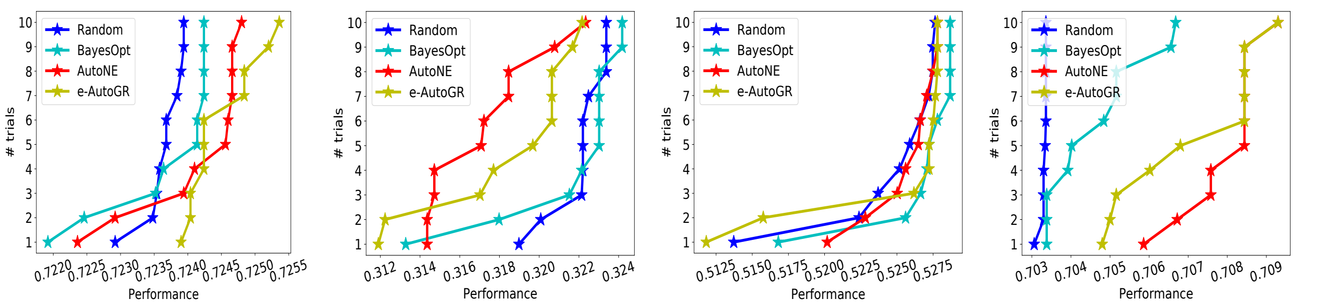
Node Classification. Figure 1 (a), (b) and (c) present the node classification performance of running AROPE on Pubmed, BlogCatalog and Wikipedia. It is easy to observe that our proposed e-AutoGR model beats all other methods in 8 out of 10 trials and finds the best optimal hyperparameter configuration after the final trial on Pubmed. Figure 2 (b) and (c) depict the node classification performance of executing DeepWalk on BlogCatalog and Wikipedia. It is observable that e-AutoGR performs almost as good as the best state-of-the-art method after 10 rounds of executing DeepWalk on BlogCatalog. We also observe from Figure 2 (a) that the proposed e-AutoGR framework running GCN achieves a comparable performance with other baselines after the last trial on Pubmed.

Link Prediction. Figure 1 (d) and Figure 2 (d) demonstrate the link prediction performance of running AROPE and DeepWalk on Wikipedia respectively. The proposed e-AutoGR running AROPE performs the best in 5 out of 10 trials and beats all other comparative partners in the end. When selecting DeepWalk as the executing graph representation algorithm, our e-AutoGR model also outperforms the best baseline method after the final trial.

Hyperparameter Importance. The goal of e-AutoGR is to answer questions such as *what is the importance of each hyperparameter, how it contributes to the model performance and why some certain values of hyperparameters are chosen in the next round of evaluation* for automated graph representation learning while maintaining a comparable or even better model performance. Figure 3 shows the importance explanation of different hyperparameters learned by e-AutoGR. We observe that for AROPE, *1st order proximity* is the most important hyperparameter for both node classification and link prediction on Wikipedia, which makes much sense because higher order proximity in AROPE has a larger uncertainty range, indicating that we should focus more on tuning lower order proximity. For Deepwalk, both tasks on Wikipedia demonstrate *number of walks* and *window size* are more important than *length of each walk*. We optimize five hyperparameters in GCN, out of which *dropout* is the most important hyperparameter influencing the performance and *weight decay* is the least important hyperparameter. These results are also reasonable because *dropout* effectively avoids model overfitting, which is very important in practice, while *weight decay* is a hyperparameter practically less tuned.

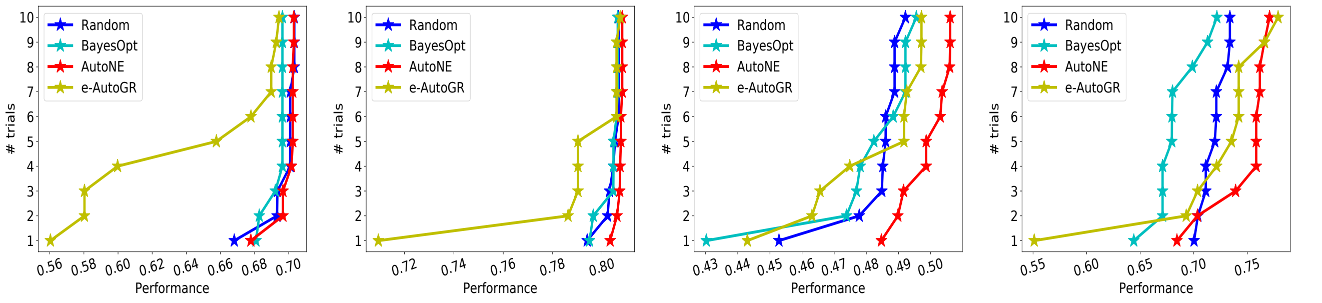
Therefore, we conclude that the proposed e-AutoGR framework can indeed learn reasonable hyperparameter importance and provide insights in hyperparameter optimization for explainable automated graph representation learning.

Explainability v.s. Performance. It is generally accepted by the community that there is a trade-off between model explainability and model performance, increasing model explainability tends to result in decreasing model performance. In this work, we try to increase the model explainability as well as keeping or improving model performance, which can



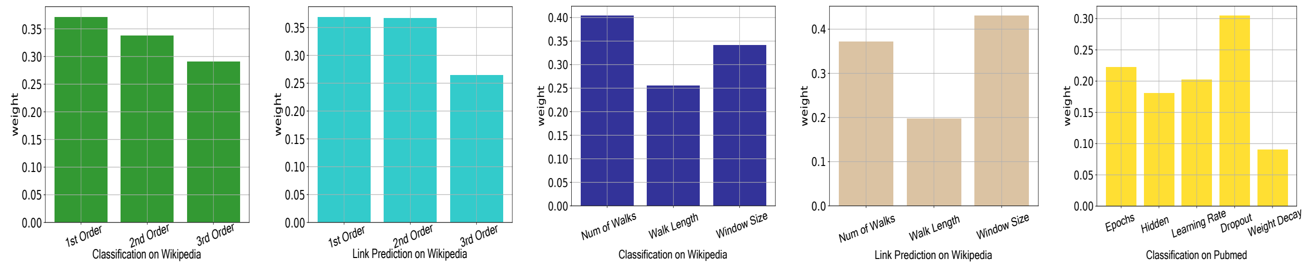
(a) (b) (c) (d)

Figure 1. Number of trials to execute algorithm R on graph G, i.e., T, vs. performance. The graph representation algorithm R is AROPE. (a) Classification on Pubmed. (b) Classification on BlogCatalog. (c) Classification on Wikipedia. (d) Link Prediction on Wikipedia. Performances are measured based on the average values of five executions for each algorithm.



(a) (b) (c) (d)

Figure 2. Number of trials executing algorithm R, i.e., T, vs. performance. The graph representation algorithm R is GCN for (a) and DeepWalk for (b)(c)(d). (a) Classification on Pubmed. (b) Classification on BlogCatalog. (c) Classification on Wikipedia. (d) Link Prediction on Wikipedia. Performances are measured based on the average values of five executions for each algorithm.



(a) AROPE (b) AROPE (c) DeepWalk (d) DeepWalk (f) GCN

Figure 3. Explaining hyperparameter importance in affecting performance.

be demonstrated from our experiments that the proposed model achieves comparable performance with black-box baselines in 3 (Figure 1(c), Figure 2(a)(b)) out of 8 cases and even outperforms baselines in 3 (Figure 1(a)(d), Figure 2(d)) out of 8 cases. For those results not as good as baselines,one possible explanation is that the random initialization process in our setting for the original large-scale graph may lead to bad initial values of the hyperparameters, which can be alleviated by utilizing the best hyperparameter settings in previous trials on sampled subgraphs.

5. Conclusion

In this paper, we explore explainable automated graph representation learning with hyperparameter importance. We propose a hyperparameter decorrelation algorithm to achieve hyperparameter importance explanation such that our explainable e-AutoGR framework demonstrates a much higher degree of rationality and explainability compared with existing literature. Extensive experiments on real-world datasets validate the effectiveness of the proposed model in providing rational insights in learning more accurate hyperparameter importance when achieving superior performance. Further improving the explainability in automated graph representation learning deserves future investigations.

Acknowledgements

This research is supported by the National Key Research and Development Program of China (No.2020AAA0106300, No.2020AAA0107800, No.2018AAA0102000) and National Natural Science Foundation of China No.62050110.

References

Bergstra, J. and Bengio, Y. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(Feb):281–305, 2012.

Bonner, S., Brennan, J., Theodoropoulos, G., Kureshi, I., and McGough, A. Efficient comparison of massive graphs through the use of'graph fingerprints'. 2016.

Cai, H., Zhu, L., and Han, S. Proxylessnas: Direct neural architecture search on target task and hardware. 2018.

Cao, S., Lu, W., and Xu, Q. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM international on conference on information and knowledge management*, pp. 891–900, 2015.

Chinchor, N. The statistical significance of the muc-4 results. In *Proceedings of the 4th conference on Message understanding*, pp. 30–50. Association for Computational Linguistics, 1992.

Escalante, H. J., Montes, M., and Sucar, L. E. Particle swarm model selection. *Journal of Machine Learning Research*, 10(Feb):405–440, 2009.

Escalante, H. J., Montes, M., and Sucar, E. Ensemble particle swarm model selection. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. IEEE, 2010.

Fang, M., Li, Y., and Cohn, T. Learning how to active learn: A deep reinforcement learning approach. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 595–605, 2017.

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., and Hutter, F. Efficient and robust automated machine learning. In *Advances in neural information processing systems*, pp. 2962–2970, 2015.

Fortunato, S. Community detection in graphs. *Physics reports*, 486(3-5):75–174, 2010.

Golovin, D., Solnik, B., Moitra, S., Kochanski, G., Karro, J., and Sculley, D. Google vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1487–1495, 2017.

Grover, A. and Leskovec, J. node2vec: Scalable feature learning for networks. In *SIGKDD*. ACM, 2016.

Guo, Z., Zhang, X., Mu, H., Heng, W., Liu, Z., Wei, Y., and Sun, J. Single path one-shot neural architecture search with uniform sampling. *arXiv*, 2019.

Hamilton, W., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. In *NeurIPS*, 2017a.

Hamilton, W. L., Ying, R., and Leskovec, J. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017b.

Hansen, B. E. Lecture notes on nonparametrics. *Lecture notes*, 2009.

Hassanpour, N. and Greiner, R. Learning disentangled representations for counterfactual regression. In *International Conference on Learning Representations*, 2019.

Hutter, F., Hoos, H. H., and Leyton-Brown, K. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pp. 507–523. Springer, 2011.

Hutter, F., Hoos, H., and Leyton-Brown, K. An efficient approach for assessing hyperparameter importance. In *International conference on machine learning*, pp. 754–762. PMLR, 2014.

Hutter, F., Kotthoff, L., and Vanschoren, J. *Automated Machine Learning*. Springer, 2019.

Kanter, J. M. and Veeramachaneni, K. Deep feature synthesis: Towards automating data science endeavors. In *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pp. 1–10. IEEE, 2015.

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

Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.-J., Fei-Fei, L., Yuille, A., Huang, J., and Murphy, K. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 19–34, 2018.

Liu, H., Simonyan, K., and Yang, Y. Darts: Differentiable architecture search. In *The International Conference on Learning Representations (ICLR)*, 2019.

Luo, G. A review of automatic selection methods for machine learning algorithms and hyper-parameter values. *Network Modeling Analysis in Health Informatics and Bioinformatics*, 5(1):18, 2016.

Ma, J., Cui, P., Kuang, K., Wang, X., and Zhu, W. Disentangled graph convolutional networks. In *International Conference on Machine Learning*, pp. 4212–4221, 2019.

Mantovani, R. G., Horváth, T., Cerri, R., Vanschoren, J., and de Carvalho, A. C. Hyper-parameter tuning of a decision tree induction algorithm. In *2016 5th Brazilian Conference on Intelligent Systems (BRACIS)*, pp. 37–42. IEEE, 2016.

Myerson, J., Green, L., and Warusawitharana, M. Area under the curve as a measure of discounting. *Journal of the experimental analysis of behavior*, 76(2):235–243, 2001.

Nakatsukasa, Y. Absolute and relative weyl theorems for generalized eigenvalue problems. *Linear Algebra and its Applications*, 432(1):242–248, 2010.

Ou, M., Cui, P., Pei, J., Zhang, Z., and Zhu, W. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 1105–1114, 2016.

Perozzi, B., Al-Rfou, R., and Skiena, S. Deepwalk: Online learning of social representations. In *SIGKDD*. ACM, 2014.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. Regularized evolution for image classifier architecture search. In *AAAI*, volume 33, pp. 4780–4789, 2019.

Roweis, S. T. and Saul, L. K. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500): 2323–2326, 2000.

Sanders, S. and Giraud-Carrier, C. Informing the use of hyperparameter optimization through metalearning. In *2017 IEEE International Conference on Data Mining (ICDM)*, pp. 1051–1056. IEEE, 2017.

Sasaki, Y. The truth of the f-measure. *F-measure-YS-26Oct07.pdf*, 2007.

Snoek, J., Larochelle, H., and Adams, R. P. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pp. 2951–2959, 2012.

Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., and Mei, Q. Line: Large-scale information network embedding. In *WWW*. International World Wide Web Conferences Steering Committee, 2015.

Tenenbaum, J. B., De Silva, V., and Langford, J. C. A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500):2319–2323, 2000.

Thornton, C., Hutter, F., Hoos, H. H., and Leyton-Brown, K. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 847–855, 2013.

Tu, K., Cui, P., Wang, X., Wang, F., and Zhu, W. Structural deep embedding for hyper-networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Tu, K., Ma, J., Cui, P., Pei, J., and Zhu, W. Autone: Hyperparameter optimization for massive network embedding. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 216–225, 2019.

Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Wang, D., Cui, P., and Zhu, W. Structural deep network embedding. In *SIGKDD*. ACM, 2016.

Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., Tian, Y., Vajda, P., Jia, Y., and Keutzer, K. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *CVPR*, pp. 10734–10742, 2019.

Yao, Q., Wang, M., Chen, Y., Dai, W., Yi-Qi, H., Yu-Feng, L., Wei-Wei, T., Qiang, Y., and Yang, Y. Taking human out of learning applications: A survey on automated machine learning. *arXiv preprint arXiv:1810.13306*, 2018.

You, J., Ying, R., Ren, X., Hamilton, W. L., and Leskovec, J. Graphrnn: A deep generative model for graphs. *arXiv preprint arXiv:1802.08773*, 2018.

Zhang, Z., Cui, P., Wang, X., Pei, J., Yao, X., and Zhu, W. Arbitrary-order proximity preserved network embedding. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2778–2786, 2018.

Zhu, D., Cui, P., Wang, D., and Zhu, W. Deep variational network embedding in wasserstein space. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2827–2836, 2018.

Zhu, W., Wang, X., and Cui, P. Deep learning for learning graph representations. In *Deep Learning: Concepts and Architectures*, pp. 169–210. Springer, 2020.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. In *The International Conference on Learning Representations (ICLR)*, 2017.