AutoNE: Hyperparameter Optimization for Massive Network Embedding

Ke Tu*
Tsinghua University
tuk15@mails.tsinghua.edu.cn

Jianxin Ma
Tsinghua University
majx13fromthu@gmail.com

Peng Cui
Tsinghua University
cuip@tsinghua.edu.cn

Jian Pei
Simon Fraser University and JD.com
jpei@cs.sfu.ca

Wenwu Zhu
Tsinghua University
wwzhu@tsinghua.edu.cn

ABSTRACT

Network embedding (NE) aims to embed the nodes of a network into a vector space, and serves as the bridge between machine learning and network data. Despite their widespread success, NE algorithms typically contain a large number of hyperparameters for preserving the various network properties, which must be carefully decided in order to achieve satisfactory performance. Though automated machine learning (AutoML) has achieved promising results when applied to many types of data such as images and texts, network data pose great challenges to AutoML and remain largely ignored by the literature of AutoML. The biggest obstacle is the massive scale of real-world networks, along with the coupled node relationships that make any straightforward sampling strategy problematic. In this paper, we propose a novel framework, named AutoNE, to automatically optimize the hyperparameters of a NE algorithm on massive networks. In detail, we employ a multi-start random walk strategy to sample several small sub-networks, perform each trial of configuration selection on the sampled sub-network, and design a meta-leaner to transfer the knowledge about optimal hyperparameters from the sub-networks to the original massive network. The transferred meta-knowledge greatly reduces the number of trials required when predicting the optimal hyperparameters for the original network. Extensive experiments demonstrate that our framework can significantly outperform the existing methods, in that it needs less time and fewer trials to find the optimal hyperparameters.

KEYWORDS

Network Representation Learning; Network Embedding; Automated Machine Learning; Meta-Learning; Hyperparameter Optimization

*Beijing National Research Center for Information Science and Technology (BNRist)

KDD '19, August 4–8, 2019, Anchorage, AK, USA

ACM ISBN 978-1-4503-6201-6/19/08…$15.00
https://doi.org/10.1145/3292500.3330848

ACM Reference Format:
Ke Tu, Jianxin Ma, Peng Cui, Jian Pei, and Wenwu Zhu. 2019. AutoNE: Hyperparameter Optimization for Massive Network Embedding. In *The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '19), August 4–8, 2019, Anchorage, AK, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3292500.3330848

1 INTRODUCTION

Nowadays, networks are widely used to represent complex relationships of objects, such as social networks, biology networks, etc. To process network data effectively and efficiently, many network embedding (NE) algorithms [6, 20, 28] have been proposed. NE aims to embed the nodes of a network into a low-dimensional vector space, so that the downstream applications, such as recommendation [37], node classification [14] and clustering [36], can be readily solved by applying traditional machine learning methods in the vector space. Despite the widespread success of NE, the configuration, particularly the hyperparameters, of a NE algorithm must be carefully tuned in order to achieve satisfactory performance. This tuning process relies heavily on the experience of human experts, and the issue is further aggravated by the fact that NE algorithms typically possess a large number of hyperparameters that are related with the various network properties to be preserved.

Lately, automated machine learning (AutoML) [22] has aroused great research interests from both academia and industry. It aims to ease the adoption of machine learning and reduce the reliance on human experts, by automating the various stages of machine learning, e.g., hyperparameter optimization. Many AutoML frameworks are proposed and have demonstrated their usefulness when applied to various types of data, such as images [42] and texts [7]. However, network data remain largely unexplored by the existing literature of AutoML.

The large scale of real-world networks pose great challenges to incorporating AutoML into NE. A real-world network usually contains millions or even billions of nodes and edges, while the computational cost of a NE algorithm increases proportionally to the network size. As a result, it is unrealistic to automatically search for the optimal configuration by executing the target NE algorithm on such a massive network for a large number of times. In this paper, we suggest to perform each trial of configuration selection on a sampled sub-network, and to find a way to transfer the knowledge about optimal hyperparameters from the sub-networks to the original massive network. A straightforward way to conduct

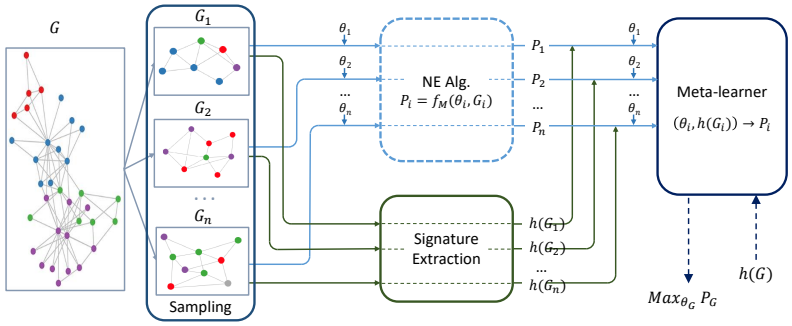


Figure 1: The transfer process of AutoNE. It involves three modules: the sampling module, the signature extraction module, and the meta-learning module. $h(G_i)$ is the signature of G_i, and P_i is the performance achieved on G_i using configuration θ_i.

configuration selection on sampled sub-networks still faces the following challenges:

(1) **Transferability**: The optimal configuration for a sampled sub-network is in general not the optimal configuration for the original massive network, because sampling breaks the coupled relationships among nodes and inevitably introduces notable bias. This is in contrast to unstructured data such as images, where it is easy to sample a subset that follows the identical distribution as the original dataset. Predicting the optimal configuration for the original network thus requires mining transferable knowledge from the sub-networks.

(2) **Heterogeneity**: A network usually consists of several highly heterogeneous components, e.g., communities, many of which may be lost after sampling. If the sampling procedure is not carefully designed, an excessive number of samples may be required to cover all the information that is essential for deciding the optimal configuration for the original network. Moreover, when assessing the transferability of a sub-network based on its similarity to the original network, the measurement must be well aware of the heterogeneity.

To address the above challenges, we propose a novel framework, named AutoNE, for automatically deciding the optimal hyperparameter configuration of a NE algorithm. Our framework consists of three major modules: the sampling module, the signature extraction module, and the meta-learning module. The sampling module samples several small sub-networks from the original large-scale network. It employs a multi-start random walk strategy, where the multiple starting points are chosen to preserve the heterogeneity. The signature extraction module then produces a signature for each network (including both the sub-networks and the original network), to facilitate the measurement of similarity between two networks. It extracts the signature based on the Laplacian spectrum, which captures the network properties comprehensively and is capable of differentiating between the heterogeneous components. Finally, the meta-learning module trains a Gaussian process meta-learner on the sampled sub-networks to distill transferable meta-knowledge about optimal hyperparameter configurations. Specifically, the meta-learner is trained to estimate the underlying function that maps a hyperparameter configuration and a network signature to the performance of the target NE algorithm. To predict the optimal hyperparameters for the original network, the meta-learner fixes one of the arguments as the signature of the original network, and optimizes the other argument (i.e., the hyperparameter configuration), to maximize the output of the function (i.e., the estimated performance). The transfer process is shown in Figure 1. Thanks to the meta-knowledge transferred from the sampled sub-networks, our framework can achieve superior performance with only a minimal number of trials on the original large-scale network.

To summarize, the contributions of our papers are as follows:

- We investigate the pressing problem of incorporating AutoML into NE, and propose AutoNE, a novel framework that automates hyperparameter optimization for NE.
- Our framework can scale up to massive real-world networks by utilizing the meta-knowledge transferred from sampled sub-networks. In particular, the sampling module and the signature extraction module are designed with the **Heterogeneity** issue in mind. And the **Transferability** challenge is addressed with a sophisticated meta-learning module.
- Experimental results on real-world networks demonstrate both the effectiveness and the efficiency of our framework for several representative NE algorithms.

The rest of the paper is organized as follows. In Section 2, we briefly review the related works. We then formally define the research problem and present the details of the proposed framework in Section 3. We report the experimental results in Section 4. Finally, we conclude in Section 5.

2 RELATED WORK

2.1 Network Embedding

Network embedding (NE) [6, 10, 17, 31], which aims to preserve the node similarity in a vector space, has attracted considerable research attention in the past few years. The NE algorithms can be categorized roughly into three classes, i.e., sampling-based algorithms, factorization-based algorithms, and deep neural network-based algorithms. The sampling-based algorithms are inspired by word2vec [18]. DeepWalk [20] samples random walks to explore the structure of a network and employs word2vec after treating the

sampled walks as sentences. LINE [28] is also built on word2vec, but uses edge sampling instead of random walks. Node2vec [9] extends DeepWalk and proposes a biased random walk procedure to capture the diverse connectivity patterns in a network. The factorization-based algorithms construct a proximity matrix based on the adjacency matrix of a network, and derive node representations by decomposing the proximity matrix. For example, M-NMF [36] proposes modularized nonnegative matrix factorization to incorporate the community structure. As for the deep neural network-based algorithms [19, 32, 39, 41], SDNE [35] proposes an autoencoder to preserve the first-order and second-order proximity between nodes. And TriDNR[19] proposes a coupled deep model that incorporates the network structure, node attributes, and node labels into the learned node representations. More recently, the graph convolutional network (GCN) [14] has been proposed to process network data in an end-to-end manner. However, the need to preserve various network properties inevitably brings an excessive number of hyperparameters, and it is therefore of high demand to automate the process of hyperparameter optimization for network embedding.

2.2 Automated Machine Learning

Automated machine learning (AutoML) [22], which attempts to reduce the reliance on human assistance during the machine learning process, has emerged as an important topic in both academia and industry. AutoML targets various stages of the machine learning process, e.g., data preparation [7], feature engineering [13], model selection, neural architecture search [16, 42], and hyperparameter optimization [29]. Among these stages, the one that is most related to our work is hyperparameter optimization. Grid search and random search [2] are the two most straightforward approaches for searching for a good set of hyperparameters. Yet these two approaches do not leverage the past experience, i.e., the results of the previous trials, when deciding the hyperparameters for the next trial. To utilize the past experience and reduce the number of trials required, sequential model-based optimization (SMBO) [12] is proposed. SMBO learns a surrogate function from the past experience to approximate the unknown function that maps a set of hyperparameters to the expected performance of the hyperparameters. Bayesian optimization [26] is one of the most popular SMBO approaches. Bayesian optimization uses a Gaussian process to represent the surrogate function and decides the hyperparameters for the next trial by maximizing the expected improvement. On the other hand, hyperparameter optimization can also be viewed as a mete-learning problem [8, 34]. In this latter setting, a meta-learner extracts transferable knowledge from the hyper-parameter configurations that have been adopted in previous similar tasks, and generalizes the knowledge to predict the optimal hyperparamters when given a new task. However, the existing AutoML techniques mostly deal with image or text data, and cannot easily handle large-scale networks due to the coupled relationships among the nodes.

3 THE PROPOSED FRAMEWORK

3.1 Notations and Problem Formulation

Let $G = (V, E)$ denote a network, where V is the set of nodes and $E \subseteq V \times V$ is the set of edges. We use $G_i = (V_i, E_i)$ to denote the i^{th} sub-network sampled from the original network G. Let M be a NE algorithm. The algorithm outputs d-dimensional node representations, i.e., $M(\theta, G) \in \mathbb{R}^{|V| \times d}$, when given hyperparameter configuration θ and network $G = (V, E)$. We then use $f_M(\theta, G)$ to denote the performance of the output $M(\theta, G)$ on the validation dataset of a downstream application, e.g., node classification or link prediction. The shape of the performance function $f_M(\cdot, \cdot)$ is unknown in practice, and we will use a meta-learner to estimate it based on M's performance on the sampled sub-networks $\{G_i\}_{i>0}$.

Our goal is to find the set of hyperparameters that can achieve the optimal performance on a given network. For this purpose, we propose a novel framework named AutoNE (see Figure 1), which combines the ideas of meta-learning and Bayesian optimization. Our framework consists of three key components: the sampling module, the signature extraction module, and the meta-learning module.

3.2 The Sampling Module

To efficiently collect information about the unknown function $f_M(\cdot, \cdot)$ that maps a set of hyperparameter and a network to the expected performance of algorithm M, we will start by sampling some sub-networks from the original network.

We aim to sample a series of representative sub-networks that share similar properties with the original large-scale network. We sample each sub-network based on several random walks. And we vary the the length of each random walk to obtain sub-networks of various sizes. At each step of a random walk, we randomly select the next position v' from the neighbors of the current position v, i.e., $v' \in \mathcal{N}(v) = \{u \mid (v, u) \in E\}$, and jump to node v'. Let V_i be the nodes traversed by the random walks. The sampled sub-network $G_i = (V_i, E_i)$ is then the sub-network induced in the original network G by the sampled nodes V_i. In other words, the sub-network $G_i = (V_i, E_i)$ contains these edges:

$$E_i = \{(u, v) \mid u \in V_i \wedge v \in V_i \wedge (u, v) \in E\}, \tag{1}$$

where E contains the edges in the original large-scale network.

Moreover, to address the heterogeneity issue, i.e., to ensure that the sampled sub-network G_i preserves the diversity exhibited in the original network, the starting points of the random walks are explicitly chosen to be at the different regions of the original network. In particular, for supervised network applications such as node classification, we choose several nodes with different labels as the starting points. As for unsupervised network applications such as link prediction, the starting nodes can be chosen to be from the different communities discovered by a fast community detection algorithm, e.g., a greedy algorithm that maximizes modularity [5].

3.3 The Signature Extraction Module

The signature of a network is a vector descriptor that encodes the various properties of the whole network, typically obtained in an unsupervised manner that does not require training. By extracting the signatures of G and $\{G_i\}_{i>0}$, we can conveniently measure the similarity between two networks by comparing their signatures in the vector space. We would like the signatures to be comprehensive enough so that they capture the properties that are important for deciding the hyperparameters of a NE algorithm. Moreover, the

signatures should reflect the heterogeneous components, e.g., communities, of a network, so that the transfer process can be more aware of the sampling bias.

According to spectral graph theory [4], a large number of network properties, such as the normalized cuts [25] used by spectral clustering, are decided by the spectrum of a network, and some networks are even determined by their spectrum [33]. We therefore use NetLSD [30], a state-of-the-art method that builds on the Laplacian spectrum and preserves the community structure of a network, for signature extraction. NetLSD considers a heat diffusion process on a network and computes the heat trace at time t:

$$h_t = tr(\mathbf{H}_t) = tr(e^{-t\mathbf{L}}) = \sum_j e^{-t\lambda_j}, \tag{2}$$

where $(\mathbf{H}_t)_{ij}$ represents the amount of heat transferred from node v_i to node v_j at time t, $\mathbf{L}$ is the Laplacian matrix of the network whose signature is to be extracted, and λ_j is the j^{th} eigenvalue of the Laplacian matrix. NetLSD then outputs the heat traces at different time scales as the signature of a network, i.e., $h(G) = \{h_t\}_{t>0}$ for network G. The time complexity of NetLSD is linear with respect to the network size.

At larger time scales, the heat diffuses farther and reflects more global properties of the network. This phenomenon is closely related with the concept of high-order proximity, which is adopted by many NE algorithms [28]. Moreover, NetLSD is size-invariant, i.e. it preserves structural similarity regardless of network magnitude, which is useful for determining the hyperparameters that are hardly related with the magnitude of a network, e.g., the hyperparameters that control the importance of first-order proximity and second-order proximity. However, there might be hyperparameters that are related with the network size, e.g., the learning rate. We therefore concatenate $h(G)$ and the network size $|V|$ to form the final signature of network G.

3.4 The Meta-Learning Module

The meta-learner will collect the knowledge about the unknown performance function $f_M(\cdot,\cdot)$, by executing algorithm M on the sampled sub-networks $\{G_i\}_{i>0}$ using various hyperparameters. It will then predict the optimal hyperparameters for algorithm M on the original large-scale network G, based on the knowledge transferred from the sampled sub-networks.

3.4.1 Kernel Function. Our central assumption is that, if the signatures of two networks are similar, then with a similar set of hyperparameters, the performance of the NE algorithm will be similar on the two networks.

To begin with, we need to formally define the similarity between two sets of hyperparameters, as well as the similarity between two networks. We use two separate kernel functions for this purpose. Specifically, We define the similarity between two sets of hyperparameters as $k_\theta(\theta_1,\theta_2)$, where $k_\theta(\cdot,\cdot)$ is a Matérn 5/2 kernel function [26]. We then define the similarity between network G_1 and network G_2 as $k_g(h(G_1),h(G_2))$, where $h(G)$ is the signature of network G and $k_g(\cdot,\cdot)$ is another Matérn 5/2 kernel function. Finally, we define the similarity between (θ_1,G_1) and (θ_2,G_2) as:

$$k\left((\theta_1,G_1),(\theta_2,G_2)\right) = k_\theta\left(\theta_1,\theta_2\right)\cdot k_g\left(h(G_1),h(G_2)\right). \tag{3}$$

Compared with the other kernels such as the widely used radial basis function (RBF) kernel, the Matérn 5/2 kernel is more capable of describing a non-smooth function, and is therefore more suitable for our task where the performance function $f_M(\cdot,\cdot)$ is highly non-smooth.

3.4.2 Gaussian Process. We use a Gaussian process (GP) [23] to estimate the shape of the performance function $f_M(\cdot,\cdot)$, i.e., the mapping from hyperparameter configuration θ and network G to the expected performance $f_M(\theta,G)$. The GP can be viewed as a probabilistic distribution over the unknown function $f_M(\cdot,\cdot)$, and the distribution is updated every time we observe the value of $f_M(\theta_i,G_i)$ at a new point (θ_i,G_i).

The GP builds upon the kernel function described earlier, and does not introduce any extra parameters. Let matrix $\mathbf{X}$ be a collection of sampled points, i.e., each row of $\mathbf{X}$ is a sampled point (θ_i,G_i) that consists of a hyperparameter configuration θ_i and a network G_i. We then use matrix $K(\mathbf{X},\mathbf{X})$ to denote the similarity scores between any two sampled points, i.e., $(K(\mathbf{X},\mathbf{X}))_{ij} = k\left((\theta_i,G_i),(\theta_j,G_j)\right)$. Let $\mathbf{f}$ be a column vector consisting of the values of $f_M(\cdot,\cdot)$ at these sampled points, i.e., $(\mathbf{f})_i = f_M(\theta_i,G_i)$. The Gaussian process then assumes that for any collection of sampled points $\mathbf{X}$, the value of the performance function $f_M(\cdot,\cdot)$ at these sampled points follows the following multivariate normal distribution:

$$\mathbf{f} \mid \mathbf{X} \sim \mathcal{N}\left(\mu(\mathbf{X}), K(\mathbf{X},\mathbf{X})\right), \tag{4}$$

where $\mu(\cdot)$ is a mean function. We set the mean function to a constant zero function, i.e., $\mu(\mathbf{X}) = \mathbf{0}$, because a zero-mean GP is already expressive enough to characterize a complex, highly nonlinear function, provided that the co-variance $K(\mathbf{X},\mathbf{X})$ is based on a sophisticated kernel function such as the Matérn kernel [23].

The above assumption will allow us to predict the performance of algorithm M with a new set of hyperparameters θ_* on a new network G_*, by computing the posterior probability conditioned on the observed values of $f_M(\cdot,\cdot)$ at several sampled points. We will provide more details on this later.

3.4.3 Fitting the Gaussian Process. While the GP itself is non-parametric, the kernel used by the GP does contain parameters that can be optimized to more accurately estimate the shape of the performance function $f_M(\cdot,\cdot)$.

We therefore run the target NE algorithm M on the sampled sub-networks $\{G_i\}_{i>0}$ with some randomly selected hyperparameters $\{\theta_i\}_{i>0}$, and collect the observed performance $\mathbf{f} = \{f_M(\theta_i,G_i)\}_{i>0}$ at these sampled points $\mathbf{X} = \{(\theta_i,G_i)\}_{i>0}$. We then decide the parameters of the kernel function $k(\cdot,\cdot)$ by maximizing the marginal likelihood $p(\mathbf{f} \mid \mathbf{X})$ of the observed data. Under the assumption of the Gaussian process, the log likelihood can be expressed as follows:

$$\ln p(\mathbf{f} \mid \mathbf{X}) = -\frac{1}{2}\mathbf{f}^\top K(\mathbf{X},\mathbf{X})^{-1}\mathbf{f} - \frac{1}{2}\ln\det(K(\mathbf{X},\mathbf{X})) + constant. \tag{5}$$

We use L-BFGS-B [40], a quasi-Newton method, to maximize it.

3.4.4 Predicting the Optimal Hyperparameters. Here we will show how the GP can efficiently estimate the expected performance $f_M(\theta_*,G_*)$ when given a new set of hyperparameters θ_* and a new network G_*, and how we can predict the optimal hyperparameter configuration θ_* when given a new network G_*.

Given a new test point $\mathbf{x}_* = (\theta_*, G_*)$ and the observed values $\mathbf{f}$ at the existing sampled points $\mathbf{X}$, the assumption of the GP implies that $f_M(\theta_*, G_*)$ and $\mathbf{f}$ follow a joint normal distribution:

$$\begin{bmatrix} \mathbf{f} \\ f_M(\theta_*, G_*) \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} K(\mathbf{X}, \mathbf{X}) & K(\mathbf{X}, \mathbf{x}_*) \\ K(\mathbf{x}_*, \mathbf{X}) & K(\mathbf{x}_*, \mathbf{x}_*) \end{bmatrix}\right), \tag{6}$$

where $K(\mathbf{x}_*, \mathbf{x}_*) = k((\theta_*, G_*), (\theta_*, G_*))$, and $K(\mathbf{X}, \mathbf{x}_*) = K(\mathbf{x}_*, \mathbf{X})^\top$ is the similarity between the new test point and the existing sampled points, measured by the kernel function $k(\cdot, \cdot)$. As a result, it can be shown that the posterior distribution $p(f_M(\theta_*, G_*) \mid \mathbf{x}_*, \mathbf{f}, \mathbf{X})$ is a normal distribution:

$$f_M(\theta_*, G_*) \mid \mathbf{x}_*, \mathbf{f}, \mathbf{X} \sim \mathcal{N}(\mu_*, \sigma_*^2), \tag{7}$$

$$\mu_* = K(\mathbf{x}_*, \mathbf{X})K(\mathbf{X}, \mathbf{X})^{-1}\mathbf{f}, \tag{8}$$

$$\sigma_*^2 = K(\mathbf{x}_*, \mathbf{x}_*) - K(\mathbf{x}_*, \mathbf{X})K(\mathbf{X}, \mathbf{X})^{-1}K(\mathbf{X}, \mathbf{x}_*). \tag{9}$$

The derivation can be found in [23]. The expected performance of algorithm M on a new network G_* using a new set of hyperparameters θ_* is therefore μ_* according to the GP.

To find the optimal hyperparameter configuration θ_* when given a new network G_*, we can treat θ_* as a parameter and vary it to maximize the potential reward. To be specific, we search for the optimal θ_* by maximizing the upper confidence bound (UCB) [1, 27]:

$$\arg\max_{\theta_*} \mu_* + \kappa\sigma_*. \tag{10}$$

Note that both μ_* and σ_* depend on θ_*. This maximization problem can be efficiently solved using a quasi-Newton method such as L-BFGS-B [40]. The constant $\kappa > 0$ controls how much risk the meta-learner is willing to take. When κ is large, the meta-learner is more willing to take risk and will actively explore regions with large variance.

3.5 The Two Phases of Hyperparameter Optimization

The hyperparameter optimization procedure consists of two phases.

During the first phase, the goal of the meta-learner is to collect information about the performance function $f_M(\cdot, \cdot)$ based on the results on the sampled sub-networks, i.e., $\{f_M(\theta_i, G_i)\}_{i>0}$. The meta-learner collects these results by sampling S sub-networks, and executing algorithm M on each sub-network for T times using different sets of hyperparameters.

During the second phase, the meta-learner will start to predict the hyperparameters that may lead to the optimal performance on the original large-scale network G. In this phase, the meta-learner makes L predictions sequentially, where the results of the previous predictions will be leveraged to improve the next prediction. Specifically, it makes each prediction θ_j according to Equation 10. After each prediction, it adds (θ_j, G) into $\mathbf{X}$, $f_M(\theta_j, G)$ into $\mathbf{f}$, and updates the parameters of the kernel function $k(\cdot, \cdot)$ before making the next prediction. The optimal hyperparameter configuration is then chosen from the L predictions.

The pseudocode of the whole hyperparameter optimization procedure is listed in Algorithm 1. The overall time complexity is $O(L|E|)$, which is mostly due to the need to execute algorithm M on the original network $G = (V, E)$ for L times during the second phase. The time complexity of the first phase is negligible, because the sampled sub-networks are far smaller than the original network, i.e., $|E_i| \ll |E|$. In fact, the total execution time of the first phase is usually less than or on par with that of executing algorithm M for one time on the original network. The overall time complexity of the Gaussian process is $O((ST)^3 L)$, which is again negligible, since $(ST)^3 \ll |E|$. Note that L is much smaller than the number of trials required by the existing hyperparameter optimization techniques, because our meta-learner additionally learns to collect transferable information from the sampled sub-networks.

Algorithm 1 *Automated Hyperparameter Optimization for Network Embedding* (AutoNE)

Input: Network G; Network embedding algorithm M.
Output: The optimal hyperparameter configuration θ_{opt}.

```
/* Phase I */
Sample S sub-networks from G according to Section 3.2. Each sampled sub-network will be reused for T times. We therefore use {G_i}_{i=1}^{ST} to denote the ST sub-networks.
X = {}, f = {}.
for sub-network i ← 1, 2, ..., ST do
    Compute the signature of G_i according to Section 3.3. The signature will be used by the kernel function k(·,·).
    Select a hyperparameter configuration θ_i randomly.
    Run algorithm M on sub-network G_i using θ_i, and record the performance f_M(θ_i, G_i).
    X ← X ∪ {(θ_i, G_i)}, f ← f ∪ {f_M(θ_i, G_i)}.
end for
Update the kernel parameters by maximizing Equation 5.

/* Phase II */
Compute the signature of network G according to Section 3.3.
for trial j ← ST + 1, ST + 2, ..., ST + L do
    Obtain a hyperparameter configuration θ_j that may achieve the optimal performance on G according to Equation 10.
    Run algorithm M on the input network G using θ_j, and record the performance f_M(θ_j, G).
    X ← X ∪ {(θ_j, G)}, f ← f ∪ {f_M(θ_j, G)}.
    Update the kernel parameters by maximizing Equation 5.
end for
return θ_opt = arg max_{θ_j : ST+1 ≤ j ≤ ST+L} f_M(θ_j, G).
```

4 EXPERIMENTS

In this section, we empirically analyze the proposed framework. Network embedding (NE) algorithms can be roughly categorized into three classes, including sampling-based algorithms, factorization-based algorithms, and deep neural network-based algorithms. We therefore assess the efficacy and efficiency of our framework with one representative algorithm from each class. Specifically, we optimize the hyperparameters of the NE algorithms on two common network applications, including link prediction and node classification.

Note that in order to fairly compare our framework with the baselines, we select small- or moderate-scale, rather than large-scale, network datasets in Subsection 4.2–4.4, wherein the baselines

can work well. Furthermore, we validate our framework in a large-scale network in Subsection 4.5, where all baselines cannot report reasonable results.

4.1 Baselines and Experiment Settings

In order to evaluate the effectiveness of our proposed framework, we compare our framework with two widely adopted hyperparameter optimization strategies:

- Random search [2]: Random search is one of the most commonly used strategies for hyperparameter optimization. Random search is sufficient to find the optimal solution as long as the time budget is large enough, though it does not leverage the feedback provided by the previous trials. Another commonly used strategy is grid search. However, as demonstrated by [2], random search is favored over grid search in most scenarios, because random search can explore larger configuration space more efficiently and find better solutions faster. Therefore, we use random search as one of the baselines in our experiments.
- Bayesian optimization (BayesOpt) [26]: Bayesian optimization uses a Gaussian process to serve as a surrogate of the expensive evaluation process and decides the hyperparameters for the next trial by maximizing the probability of improvement. Many AutoML frameworks incorporate Bayesian optimization as the standard method for hyperparameter optimization. Bayesian optimization performs each trial on the original data as many existing techniques for hyperparameter optimization do, which makes it inefficient at handling large-scale networks.

We set L, the number of times we are allowed to execute a NLR algorithm on the original network, to ten. Additionally, the number of sampled sub-networks S is set to five, and each sub-network is tested with $T = 5$ sets of hyperparameters. The number of nodes in each sampled sub-network is randomly selected within the range from $5\%|V|$ to $20\%|V|$. Each sub-network is sampled based on five concurrent random walks, and the random walks are stopped once the sub-network reaches the specified size.

To measure the performance of each target NE algorithm, we set up two tasks, i.e., link prediction and node classification. For link prediction, we randomly hide 20% of the edges and train the NE algorithm on the remaining part. After training, we obtain the node representations from the algorithm's output and use them to predict the held-out links based on the inner product of the learned node representations. We use the area under the curve (AUC) [11] as the evaluation metric for link prediction. For node classification, we use the learned node representations to train a logistic regression classifier. The Micro-F1 score is used as the evaluation metric for node classification.

4.2 Sampling-Based NE

Most of the sampling-based algorithms are inspired by word2vec [18]. Many algorithms from this category use random walks to explore the structure of a network. The most important hyperparameters of these NE algorithms are the number of random walks to start at each node, the length of each random walk, and the window size of the skip-gram model used by word2vec. We choose DeepWalk [20], the most representative sampling-based NE algorithm, as the algorithm to be tuned in this section, and optimize the said three hyperparameters. Two real-world networks, BlogCatalog and Wikipedia, are used for evaluating the performance of AutoNE. BlogCatalog is a social network with 10,312 nodes, 333,983 edges, and 39 categories. Wikipedia is a co-occurrence network of words appearing in the first 10^9 bytes of the English Wikipedia dump and it contains 4,777 nodes, 184,812 edges and 40 labels.

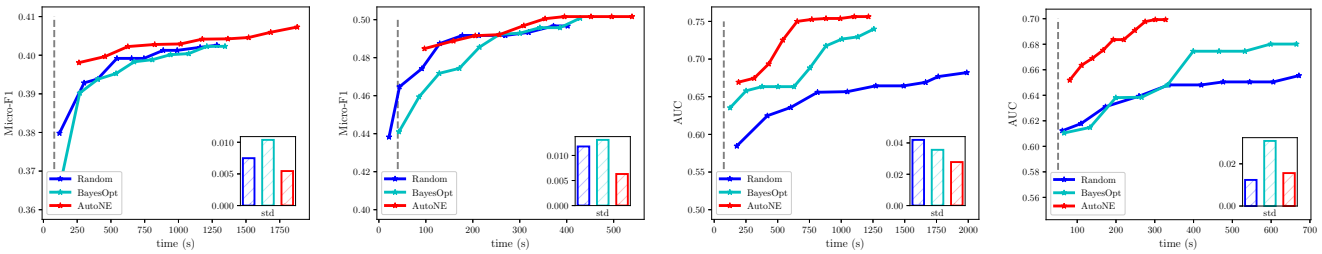
We repeat our experiments for five times for each task with the same setting, and report the mean along with the standard deviation. We measure the performance of each hyperparameter optimization method in terms of two metrics: (1) the performance achieved by each method within various time thresholds, and (2) the number of trials, i.e., how many times the NE algorithm is executed on the original network, required by each method to find a hyperparameter configuration that can reach a certain performance threshold. We report the results corresponding to these two metrics in Figure 2 and Figure 3, respectively.

It can be concluded from Figure 2 and Figure 3 that our proposed framework outperforms the baselines significantly and consistently. Note that our framework takes slightly longer time to finish its first trial than the baselines, because our framework needs to collect transferable information from the sampled sub-networks before it can start its first trial. However, the performance achieved by the first few trials of our framework is much better, because our framework can benefit from the knowledge transferred from the small sub-networks. Moreover, the standard deviation of our framework is smaller in most cases, which demonstrates the stability of our framework. On the other hand, we can see from Figure 3 that our framework takes much fewer trials to find a good hyperparameter configuration, which demonstrates that our framework is more capable of handling large-scale networks on a limited time budget.

4.3 Factorization-Based NE

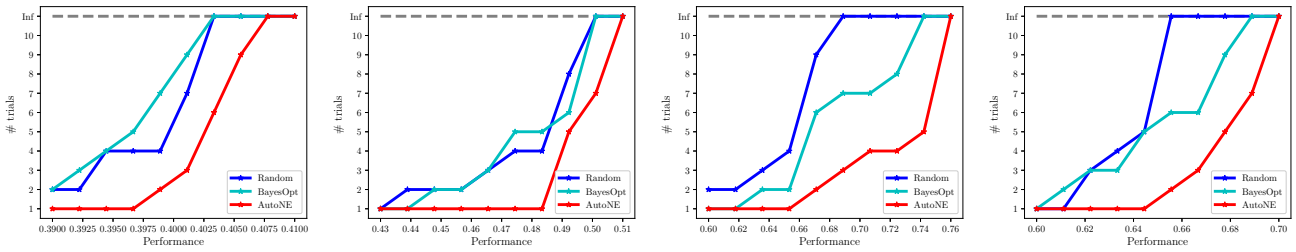
The factorization-based NE algorithms obtain node representations by computing the low-rank factorization of a proximity matrix [21]. The most typical way to construct the proximity matrix is to utilize the high-order relationships between two nodes [3, 38]. In this subsection, we choose AROPE [38], a NE algorithm that supports constructing an arbitrary-order proximity matrix, as the target to be tuned. Specifically, the hyperparameters we aim to tune are the weights of the different orders. We use the same experimental setting as Subsection 4.2 here.

We report the performance achieved by each method within various time thresholds in Figure 4, and the number of trials required by each method to reach a certain performance threshold in Figure 5. Our framework again significantly improves upon the baselines under most scenarios. We observe that the factorization-based NE algorithms are much more stable than the sampling-based NE algorithms, e.g., the performance of AROPE falls within a a much smaller range than that of DeepWalk. On the BlogCatalog dataset, the first few trials of our framework perform slightly worse than those of BayesOpt's, which indicates that the bias introduce by sampling is not negligible. However, our framework quickly outperforms BayesOpt after a few trials, because it can learn to overcome



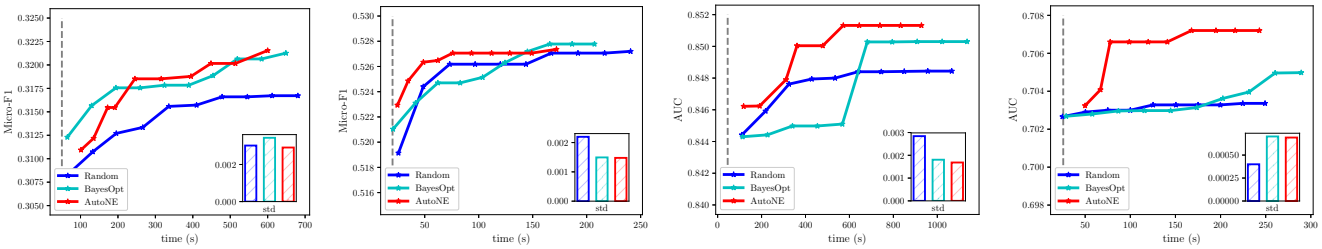
(a) Classification on BlogCatalog (b) Classification on Wikipedia (c) Link prediction on BlogCatalog (d) Link prediction on Wikipedia

Figure 2: The performance achieved by each method within various time thresholds. The NE algorithm being tuned is DeepWalk. The histogram shows the standard deviation of each method. The vertical dash line marks the time when AutoNE finishes exploring in sampled sub-networks.



(a) Classification on BlogCatalog (b) Classification on Wikipedia (c) Link prediction on BlogCatalog (d) Link prediction on Wikipedia

Figure 3: The number of trials required by each method to reach a certain performance threshold. The NE algorithm being tuned is DeepWalk. The vertical dash line marks the conjectured performance when the number of trials is unlimited.



(a) Classification on BlogCatalog (b) Classification on Wikipedia (c) Link prediction on BlogCatalog (d) Link prediction on Wikipedia

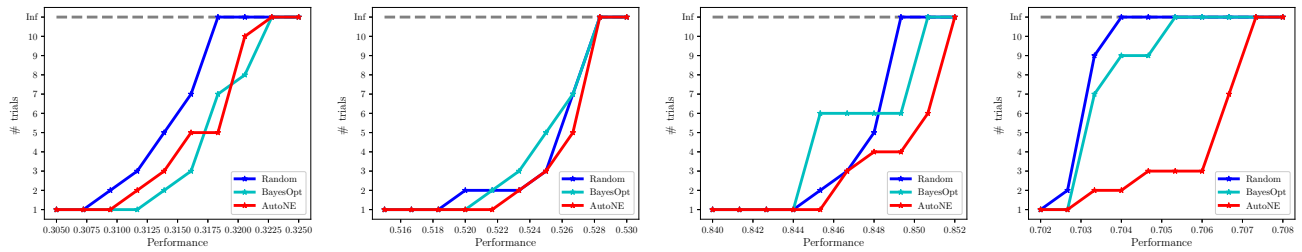
Figure 4: The performance achieved by each method within various time thresholds. The NE algorithm being tuned is AROPE.

the sampling bias after collecting sufficient data on both the sub-networks and the original network.

4.4 Deep Neural Network-Based NE

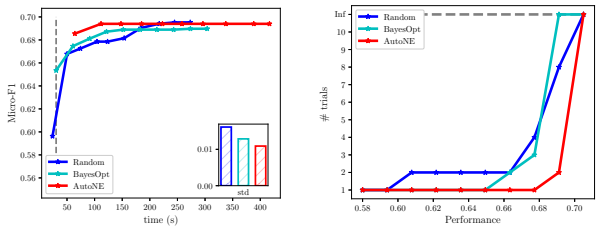
Deep neural network-based algorithms usually contain a large number of hyperparameters that have a strong influence on the performance. We choose the graph convolutional network (GCN) [14], a representative end-to-end algorithm for network data, as the target to be tuned. Specifically, the five hyperparameters we aim to tune include the learning rate, the size of each hidden layer, the number of training epochs, the dropout rate and the weight decay. The datasets used in the previous experiments do not contain node features, which are required by GCN. We therefore conduct our experiments on the Pubmed [24] dataset instead, which is a citation network that contains 19,717 nodes, 44,338 edges, 500-dimensional node features, and three classes. We sample 80% of the node labels for training. Due to the space limit, we only report the results on the node classification task here.

The results are shown in Figure 6. We can see that our framework can achieve the optimal performance in a much shorter time, which again demonstrates the effectiveness of our framework.



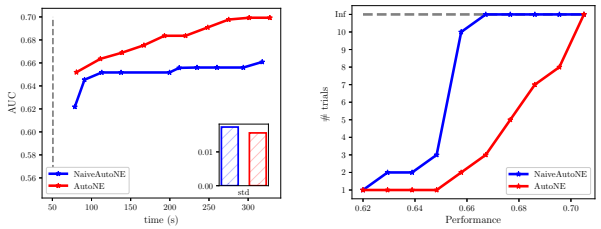
(a) Classification on BlogCatalog (b) Classification on Wikipedia (c) Link prediction on BlogCatalog (d) Link prediction on Wikipedia

Figure 5: The number of trials required by each method to reach a certain performance threshold. The NE algorithm being tuned is AROPE.



(a) The performance achieved by each method within various time thresholds. (b) The number of trials required to reach a certain performance threshold.

Figure 6: Node classification on Pubmed. The NE algorithm being tuned is GCN.



(a) The performance achieved within various time thresholds. (b) The number of trials required to reach a certain performance threshold.

Figure 7: The importance of transferring knowledge while being aware of the sampling bias.

Table 1: Results on a massive network with around thirty million edges, where we can only afford to run a NE algorithm on the whole network for a few times.

Method	Trial 1		Trial 2		Trial 3	
	AUC	Time(s)	AUC	Time(s)	AUC	Time(s)
AutoNE	0.717	1067.9	0.726	1856.2	**0.769**	2641.9
Random	0.714	698.3	0.727	1426.3	0.715	2088.6
BayesOpt	0.715	702.5	0.714	1405.1	0.727	2307.7

4.5 Analysis on a Large-Scale Network

In this subsection, we will demonstrate the ability of our framework in handling a large-scale network. Note that we can only afford to run a very small number of trials on the whole large-scale network, as executing a NE algorithm on a large-scale network is extremely time-consuming. For this purpose, we choose the Top-Cat dataset [15], which has 1,791,489 nodes and 28,511,807 edges. The number of nodes in each sampled sub-network is randomly selected within the range from 5,000 to 20,000, i.e., roughly $0.25\%|V|$ to $1\%|V|$. We tune the performance of AROPE, the fastest NE algorithm among the three we have investigated, on the link prediction task. Each hyperparameter optimization method conducts three trials on the original large-scale network.

The results are shown in Table 1. Our framework achieves significantly better performance over the baselines after merely three trials. It takes slightly more time for our framework to finish its first trial, because our framework needs extra time to collect transferable knowledge from sampled sub-networks before conducting its first trial. To overcome the bias introduced by sampling, our framework needs to collect data from not only the sampled sub-networks, but also the original network. As a result, the results of the first two trials are less significant than the result of the last trial. Overall, the results demonstrate that our framework is able to find optimal hyperparameters with only a minimal number of trials on a large-scale network.

4.6 The Necessity of Transfer

In this subsection, we will demonstrate that it is necessary to be aware of the sampling bias and have a sophisticated mechanism for transferring knowledge from the sampled sub-networks to the original network. To be specific, we compare AutoNE with a simplified variant of AutoNE, named NaiveAutoNE. NaiveAutoNE views all networks, including the sampled sub-networks and the original network, as the same one. This is achieved by replacing the signature module with a module that outputs a constant vector. We use the same setting as Figure 2 (d) and Figure 3 (d) here.

The results are shown in Figure 7. We can see that AutoNE achieves much better performance than NaiveAutoNE within a much shorter time. Moreover, NaiveAutoNE reaches a plateau

quickly while AutoNE keeps finding better hyperparameter configurations. The results demonstrate the importance of transferring knowledge while being aware of the bias introduced by sampling.

5 CONCLUSION

In this paper, we investigate the pressing problem of incorporating automated machine learning (AutoML) into network embedding (NE). To deal with large-scale real-world networks, we propose AutoNE, a novel framework for automatically optimizing the hyperparameters of a NE algorithm. Compared to the state-of-the-art AutoML methods, our framework requires less time and less trials. Extensive experiments with three representiative NE algorithms are conducted to demonstrate the effectiveness and efficiency of the proposed framework.

6 ACKNOWLEDGMENTS

We thank Chen Xuan for drawing figures and discussing. This work was supported in part by National Program on Key Basic Research Project No. 2015CB352300, National Natural Science Foundation of China Major Project No. U1611461; National Natural Science Foundation of China No. 61772304, 61521002, 61531006, U1611461; Thanks for the research fund of Tsinghua-Tencent Joint Laboratory for Internet Innovation Technology, and the Young Elite Scientist Sponsorship Program by CAST. All opinions, findings, conclusions and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

REFERENCES

[1] Peter Auer. 2002. Using Confidence Bounds for Exploitation-Exploration Tradeoffs. (2002).
[2] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research* 13, Feb (2012), 281–305.
[3] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2015. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. ACM.
[4] Fan RK Chung and Fan Chung Graham. 1997. *Spectral graph theory*. Number 92. American Mathematical Soc.
[5] Aaron Clauset, M. E. J. Newman, and Cristopher Moore. 2004. Finding community structure in very large networks. *Physical Review* (2004).
[6] Peng Cui, Xiao Wang, Jian Pei, and Wenwu Zhu. 2018. A survey on network embedding. *IEEE Transactions on Knowledge and Data Engineering* (2018).
[7] Meng Fang, Yuan Li, and Trevor Cohn. 2017. Learning how to active learn: A deep reinforcement learning approach. *arXiv preprint arXiv:1708.02383* (2017).
[8] Taciana AF Gomes, Ricardo BC Prudêncio, Carlos Soares, André LD Rossi, and André Carvalho. 2010. Combining meta-learning and search techniques to svm parameter selection. In *Neural Networks (SBRN), 2010 Eleventh Brazilian Symposium on*. IEEE, 79–84.
[9] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 855–864.
[10] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584* (2017).
[11] James A Hanley and Barbara J McNeil. 1982. The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology* 143, 1 (1982).
[12] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 2011. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*. Springer, 507–523.
[13] James Max Kanter and Kalyan Veeramachaneni. 2015. Deep feature synthesis: Towards automating data science endeavors. In *Data Science and Advanced Analytics (DSAA), 2015. 36678 2015. IEEE International Conference on*. IEEE, 1–10.
[14] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).
[15] Christine Klymko, David Gleich, and Tamara G Kolda. 2014. Using triangles to improve community detection in directed networks. *Proceedings of the ASE BigData Conference* (2014).
[16] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. 2018. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 19–34.
[17] Jianxin Ma, Peng Cui, Xiao Wang, and Wenwu Zhu. 2018. Hierarchical taxonomy aware network embedding. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 1920–1929.
[18] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.
[19] Shirui Pan, Jia Wu, Xingquan Zhu, Chengqi Zhang, and Yang Wang. 2016. Tri-party deep network representation. *Network* 11, 9 (2016), 12.
[20] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 701–710.
[21] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. 2018. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. ACM, 459–467.
[22] Yao Quanming, Wang Mengshuo, Jair Escalante Hugo, Guyon Isabelle, Hu Yi-Qi, Li Yu-Feng, Tu Wei-Wei, Yang Qiang, and Yu Yang. 2018. Taking human out of learning applications: A survey on automated machine learning. *arXiv preprint arXiv:1810.13306* (2018).
[23] Carl Edward Rasmussen. 2004. Gaussian processes in machine learning. In *Advanced lectures on machine learning*. Springer, 63–71.
[24] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. 2008. Collective classification in network data. *AI magazine* 29, 3 (2008), 93.
[25] Jianbo Shi and Jitendra Malik. 2000. Normalized cuts and image segmentation. *PAMI* (2000).
[26] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. 2012. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*. 2951–2959.
[27] Niranjan Srinivas, Andreas Krause, Sham Kakade, and Matthias Seeger. 2010. Gaussian process optimization in the bandit setting: No regret and experimental design. In *In Proceedings of the 27th International Conference on Machine Learning*.
[28] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 1067–1077.
[29] Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 2013. 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*. ACM, 847–855.
[30] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, Alex Bronstein, and Emmanuel Müller. 2018. NetLSD: Hearing the Shape of a Graph. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*.
[31] Ke Tu, Peng Cui, Xiao Wang, Fei Wang, and Wenwu Zhu. 2018. Structural Deep Embedding for Hyper-Networks. In *Thirty-Second AAAI Conference on Artificial Intelligence, 2018*.
[32] Ke Tu, Peng Cui, Xiao Wang, Philip S Yu, and Wenwu Zhu. 2018. Deep Recursive Network Embedding with Regular Equivalence. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM.
[33] Edwin R. van Dam and Willem H. Haemers. 2003. Which graphs are determined by their spectrum? *Linear Algebra Application* (2003).
[34] Joaquin Vanschoren. 2018. Meta-learning: A survey. *arXiv preprint arXiv:1810.03548* (2018).
[35] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1225–1234.
[36] Xiao Wang, Peng Cui, Jing Wang, Jian Pei, Wenwu Zhu, and Shiqiang Yang. 2017. Community Preserving Network Embedding.. In *AAAI*. 203–209.
[37] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. *arXiv preprint arXiv:1806.01973* (2018).
[38] Ziwei Zhang, Peng Cui, Xiao Wang, Jian Pei, Xuanrong Yao, and Wenwu Zhu. 2018. Arbitrary-order proximity preserved network embedding. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2778–2786.
[39] Ziwei Zhang, Peng Cui, and Wenwu Zhu. 2018. Deep Learning on Graphs: A Survey. *arXiv preprint arXiv:1812.04202* (2018).
[40] Ciyou Zhu, Richard H Byrd, Peihuang Lu, and Jorge Nocedal. 1997. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software (TOMS)* 23, 4 (1997), 550–560.
[41] Dingyuan Zhu, Peng Cui, Daixin Wang, and Wenwu Zhu. 2018. Deep variational network embedding in wasserstein space. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM.
[42] Barret Zoph and Quoc V. Le. 2017. Neural architecture search with reinforcement learning. In *Proceedings of ICLR 2017*.

A SUPPLEMENT

A.1 Hyperparameter Search Space

A hyperparameter optimization method typically searches for the optimal hyperparameters with constraints on the hyperparameters. We specify the lower bound and the upper bound for each hyperparameter as follows:

- DeepWalk: the number of random walks starting from each node is selected within the range from 2 to 20; the length of each random walk is selected within the range from 2 to 80; the window size is selected within the range from 2 to 20.
- AROPE: the weights of second-, third-, and fourth-order proximity are selected within the range from 0.0001 to 1.0.
- GCN: the number of training epochs is selected within the range from 2 to 300; the number of neurons in each hidden layer is selected within the range from 2 to 64; the learning rate is selected within the range from 0.0001 to 0.1; the dropout rate is selected within the range from 0.1 to 0.9; the weight decay, i.e., L2 regularization, is selected within the range from 0.00001 to 0.001.

A.2 Hardware Configuration and Software Versions

All experiments are conducted with the following setting:

- Operating system: Ubuntu 18.04.1 LTS
- CPU: Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz
- RAM: DDR4 1TB
- GPU: GeForce GTX Titan X
- Software versions: Python 3.6; NumPy 1.15.4; SciPy 1.2.0; NetworkX 2.2; scikit-learn 0.20.0; TensorFlow 1.11

GCN is executed on the GPU, while all the other experiments are conducted on the CPU.

A.3 Baselines and Datasets

The publicly available implementations of the baselines and the network representation learning algorithms can be found at the following URLs:

- Bayesian Optimization: https://github.com/fmfn/BayesianOptimization
- DeepWalk: https://github.com/phanein/deepwalk
- AROPE: https://github.com/ZW-ZHANG/AROPE
- GCN: https://github.com/tkipf/gcn

The datasets used in this paper can be found at the following URLs:

- BlogCatalog: http://socialcomputing.asu.edu/datasets/BlogCatalog3
- Wikipedia: https://snap.stanford.edu/node2vec/
- Pubmed: https://github.com/tkipf/gcn/tree/master/gcn/data
- TopCat: http://snap.stanford.edu/data/wiki-topcats.html