GraphEBM: Energy-based Graph Construction for Semi-Supervised Learning

Zhijie Chen[1]
Dept. of Computer Science and Engineering
Shanghai Jiao Tong University
Shanghai, China
chen-zhijie@sjtu.edu.cn

Hongtai Cao
Dept. of Computer Science
University of Illinois (UIUC)
Champaign IL, USA
hongtai2@illinois.edu

Kevin Chen-Chuan Chang
Dept. of Computer Science
University of Illinois (UIUC)
Champaign IL, USA
kcchang@illinois.edu

***Abstract*—With the rapid improvement of various techniques in graph-based semi-supervised learning, the call for higher-quality graphs becomes more intensive. However, such affinity graphs are not naturally existing in most semi-supervised learning tasks. In this paper, we propose a learning-based approach, GraphEBM, for the graph construction problem. GraphEBM is designed to address three main requirements in graph construction: 1) supporting dynamic update; 2) providing interpretable metrics; 3) tailoring to tasks. Specifically, in GraphEBM, we adopt a probabilistic view, Edge Probability Space, to model a graph construction process as constituted of events from the space. Our objective is thus to learn, by our Energy-Based Model (EBM), the latent sampling distribution. Experimental results show that our proposed GraphEBM outperforms the existing graph construction methods in improving the semi-supervised learning tasks on various datasets and it can learn global properties of a target graph only with direct local guidance.**

***Keywords*-graph construction; energy-based model; probability space; graph semi-supervised learning;**

I. Introduction

Graph-based semi-supervised learning (Graph SSL) has been proven effective in various scenarios where data points can be related for collective prediction, such as image classification [1], natural language understanding [2] and fake information detection [3]. In these applications, it has shown an outstanding capability by leveraging limited labeled data and a large amount of unlabeled data [4]. However, a key limitation in Graph SSL is that it highly depends on the quality of a given *affinity graph* that connects data points. Such a graph must satisfy the expected *smoothness*, so that close nodes have similar labels, to guarantee Graph SSL's performance.

In most cases, affinity graphs are not ready-made for raw data or hard to extract from the real world. They must first be constructed manually and this process is ineffective. In other cases, only part of the whole graph can be observed and the residual part should be deduced from the observable subgraphs. In this paper, we focus on this problem of graph inference, which aims to predict the presence or absence of edges between a set of points (as vertices of a graph).

[1]Work done in part while visiting the University of Illinois at Urbana-Champaign.

For example, in a social network like Twitter, identifying the connections among users is time-costing and sometimes infeasible– since latent relationship are often unobservable or not revealed to the public. The follower relationship is also not always reliable if we are dealing with different tasks on the network. These facts make observing the ground-truth graph directly not realistic. However, if we have access to the profile of each user, it is possible to restore the graph of interest using this side information.

We develop a scheme which, for a set of data points sampled from a latent data manifold, will construct a graph over these points as nodes with edges determined by the similarities of pairwise points, i.e., to satisfy the smoothness requirement. The scheme should satisfy the following requirements:

1) Supporting dynamic update. Most networks are changing over time, due to dynamic update, such as increase/decrease of nodes and modification of node attributes. E.g., in the Twitter network, new users are constantly added. Many users are changing their profiles and posting new tweets from time to time. Existing works do not consider these scalability issues and build the graph from scratch for updates [5]. The proposed scheme should support such dynamic updates to easily generalize to new nodes.

Insight: The scheme should be inductive, to process new nodes without retraining with existing nodes, in contrast to a transductive approach. We devise the Edge Probability Space as an inductive model (Sec. III-A) to support this property.

2) Providing interpretable metrics. To support downstream applications, a constructed graph should be interpretable. However, existing works only focus on *what* the inferred graph is. They choose the neighborhood for each node based on existing metrics [6]. In comparison, we care more about *why* an edge exists. It should provide the probability for how much likelihood of this edge, so that an application can determine how to use it.

Insight: To achieve interpretability, we develop a generative model (Sec. III-B), which defines a joint distribution on an edge and its label. This distribution can be used to measure the validity of edges.

2374-8486/20/$31.00 ©2020 IEEE
DOI 10.1109/ICDM50108.2020.00015

3) Tailoring to tasks. Most existing approaches [7] assume a fixed neighbourhood for a node, e.g., by connecting it with k nearest neighbors in terms of features. Such one-size-fits-all scheme cannot be customized for different tasks, e.g., a social-network often follows the well-know power law distribution instead of uniform degrees. Different semi-supervised methods are based on different prior knowledge on graph to take advantage of the data manifold [8]. Depending on tasks, the scheme should generate graphs of different shapes, or more specifically, with different global properties.

Insight: To customize for different tasks, the scheme should be a supervised process, guided but not only the features but also labels of nodes. We introduce the training phase (Sec III-C) which exploits supervision and we show how the global properties of graphs (Sec. III-D) are considered with direct local guidance.

Addressing these requirements, we propose a novel perspective to the graph construction problem: Based on a probabilistic view, we suggest that the edges in a graph form a distribution with unknown parameters (to be learned). Our goal is to quantify the parameters and specify the edge distribution. This paper makes the following contributions:

1) In our probabilistic view, we construct a measure space, the *Edge Probability Space*, of all edges in a graph, where each edge is assigned a measure to indicate the distribution of edges. The graph construction process is thus converted to sampling from the edge distribution. To our knowledge, we are the first to introduce this probabilistic view to the graph construction problem.

2) We propose a parametric model called *Energy-Based Model* (EBM) to learn the distribution of edges. The energy values work as a measure to indicate the validity of an edge. To tackle the sparsity and discreteness problem in an edge probability space, we propose a generator to sample edges from the EBM. We use gumbel-softmax to approximate a differentiable random walk process to induce stochasticity and facilitate back propagation. The problem is formulated as to maximize the likelihood of edges from a given graph.

3) We investigate how different settings will influence the learning process and the result of our proposed model. Extensive experiments show that using our scheme to learn the affinity graph can boost the performance on Graph SSL. We also show that it can learn different global properties of a latent graph including edge and degree distributions with only local guidance.

II. Related Works

A. Energy-Based Model

Energy-based models have a long history in machine learning [9]. They are probabilistic models that assign a scalar energy value to the random variables with energy function U. These random variables can be a composition of latent and observable variables [10]–[12] or can be input data x [13], [14]. With the assigned energy values, EBMs can be used to build probability density functions. A density function for input $x \in R^d$ is in the form of Boltzmann distribution:

$$p(x;\theta) = \frac{e^{-U(x;\theta)}}{Z(\theta)} \tag{1}$$

where θ is the model parameters to be learned. This model ensures that the data points with higher probability have lower energy values. $Z(\theta) = \int_x e^{-U(x;\theta)}$ is known as the partition function. Training EBM entails a negative sampling phase from the model distribution itself. However, obtaining samples from p_θ is of huge challenge due to the intractability of the partition function. One solution is to use Markov Chain Monte Carlo (MCMC) estimator or use Langevin function. However, these approaches are not satisfying when the data distribution is complex. Recently, neural sampler is proposed to perform fast approximate sampling. This paper also follows this route to generate negative samples.

Another trend in the development of EBM is that many deep energy models are proposed. Different from Deep Belief Nets (DBN) [15] and Deep Boltzmann Machine (DBM) [12] which involve difficult inference and learning, a deep energy model, proposed by [16], uses a deep neural network (as opposed to stacked boltzman machines in previous works) to define its energy function. In this case, the layers of the network are deterministic transformations of input data, which help check the validity of input configuration (low energy) [17]. Another advantage of using deep energy model is to enable parameterizing the energy function, which gives the model more flexibility and expressive power.

B. Graph Inference and Affinity Learning

Generally speaking, inferring graph topology from observations is a problem lacking universally-acknowledged definitions, and there are many ways to associate a topology with the observed data samples. A straight-forward but widely-used method is to compute sample correlation using a similarity function e.g. a Gaussian RBF kernel function to quantify the similarity between data samples. Historically, there have been two general approaches to learning graphs from data, one based on statistical models and the other based on physically-motivated models. The statistical perspective deems the graph structure as a generative process. Learning the graph structure is equivalent to learning a factorization of the random variables. Markov Random Fields are often applied in such perspective.

For the physically motivated models, the observation of variables is deemed as a result of an underlying physical phenomenon or process on the graph. A typical process is network diffusion, e.g. information flowing over a network or epidemic spreading between human interactions.

Readers are referred to the comprehensive work in [18] to have a closer look at the state-of-the-art approaches in graph inference. Generally speaking, our work follows the

statistical route but treats the formulation of a graph as a generative process defined on a parametric distribution.

III. Proposed Approach

A. Distribution in Edge Probability Space

We will start from defining the Edge Probability Space, the base we set our work on and we will link the graph construction problem with events in the probability space.

Suppose we have a set of nodes $\mathcal{N} = \{n_1, n_2, ..., n_N\}$, the pairwise connectivity can be denoted as E_{ij} for node n_i and n_j. An edge probability space (EPS) can be denoted as $\mathcal{S} = \{\Omega, \mathcal{F}, \mathcal{P}\}$, where Ω is called the sample space, consisting of the possible edges, i.e. $\Omega = \{E_{ij} | n_i, n_j \in \mathcal{N}\}$. The sample space Ω represents the set of possible outcomes we can sample from. $\mathcal{F}$ is called the set of events, an σ-algebra defined over Ω. Take Ξ as the set of counting measures on $\mathcal{S}$ with $\mathcal{C}$ as the smallest $\sigma-$algebra on it. A graph on $\mathcal{S}$ is a measurable map $\xi : \Omega \to \Xi$ from the probability space $\{\Omega, \mathcal{F}, \mathcal{P}\}$ to the measurable space $(\Xi, \mathcal{C})$. Every realization of a graph ξ can be defined as the combination of multiple independently sampled edges from Ω, a.k.a an event from the EPS. We denote the event as $\mathcal{E} = \{E_1, E_2, ..., E_M\}$. It can also be written as $\mathcal{E} = \sum_{i=1}^{M} \delta_{E_i}$, where δ is the Dirac measure, E_i's are random elements of $\mathcal{S}$. We will use E_i in the following sections to denote a general-purpose atomic event from the probability space. Notice that a graph can also be treated as nodes with an edge list, where duplicates of samples can be seen as the weights of the corresponding edges. So it is natural to associate an event $\mathcal{E}$ in the EPS with a graph $G = \{\mathcal{N}, \mathcal{E}\}$. The likelihood of the constructed G is then equal to the probability measure $\mathcal{P}$ of the event $\mathcal{E}$, which can be described as follows:

$$\mathcal{L}(G) = \mathcal{P}(\mathcal{E}) \propto \prod_{i=1}^{M} p_\theta(E_i) \tag{2}$$

where $p_\theta(\cdot)$ describes the probability of an atomic event E_i (an edge) with parameter θ. Our goal is to maximize the likelihood of the observed graph. Here, we derive the log likelihood of G:

$$\begin{aligned} \log \mathcal{L}(G) &= C + \sum_{i=1}^{M} \log p_\theta(E_i) \\ &= C + M\mathbb{E}_{E_i \sim p_D} \left[\log p_\theta(E_i)\right] \end{aligned} \tag{3}$$

where C is a constant associated with a combinatorial number and $\mathbb{E}$ is the estimation function. p_D is the observed distribution of the edges.

B. Energy-based Edge Density Estimation

As described above, an energy model is capable of estimating the density of a random variable from an unnormalized data distribution. We follow the classic route to define the distribution of EPS by assigning a proper energy value to each edge connection. Therefore, the edge distribution can be expressed as $p_\theta(E_{ij}) = \frac{e^{-U(E_{ij};\theta)}}{Z(\theta)}$. Then how to define the energy function $U(E_{ij}; \theta)$ is of high importance.

We suppose edge E_{ij} to be composed of vertex i and j, whose features can be denoted as $\mathbf{f}_i$ and $\mathbf{f}_j \in \mathbb{R}^d$ respectively. To guarantee the symmetry of learned affinities, we employ a siamese-structured network to extract features. It has two branches which are both multi-layer perceptrons (MLP) and share weight parameters θ.

$$\mathbf{h}_i = \mathrm{MLP}_\theta(\mathbf{f}_\mathrm{i}), \quad \mathrm{i} \in \mathcal{N} \tag{4}$$

By using the above encoding model, nodes in the feature space are projected to the embedding space, where the data points are denoted as $\mathbf{h}_i \in \mathbb{R}^m$ and the pairwise affinities can be calculated :

$$\mathbf{A}_{i,j} = f_a(\mathbf{h}_i, \mathbf{h}_j), \quad i, j \in \mathcal{N} \tag{5}$$

where the affinity matrix $\mathbf{A} \in \mathbb{R}^{+^{N \times N}}$ contains the energy values. There are several options for implementing the affinity function f_a, e.g.

$$\mathbf{A}_{i,j} = \|\mathbf{h}_i - \mathbf{h}_j\|_2 \tag{6}$$

$$\mathbf{A}_{i,j} = \exp\left(-\frac{\mathbf{h}_i^T \mathcal{M} \mathbf{h}_j}{\tau}\right) \tag{7}$$

Eq. 6 expresses the Euclidean distance between node i and node j in the embedding space. Eq. 7 is an exponential function which gives a more general metric on affinity learning. $\mathcal{M} \in \mathbb{R}^{m \times m}$ is a symmetric and trainable weight matrix of the affinity function and τ is a hyper-parameter. With the above settings, the two kinds of affinity function can both ensure all elements to be positive and lower-bounded. In the experiments, we adopt Eq. 6 since it exhibits more stability in the training process.

C. Contrastive Graph Likelihood Learning

Our training objective is to maximize the log likelihood of the observed graph, as defined in Eq. 3. Using Bolztmann Distribution latently defined by the edge density estimator, we can now derive the gradient for the objective function,

$$\begin{aligned} \bigtriangledown_\theta \log q_\theta(E_i) &= \bigtriangledown_\theta \log e^{-U(E_i;\theta)} - \bigtriangledown_\theta \log Z_\theta \\ &= -\bigtriangledown_\theta U(E_i;\theta) - \frac{1}{Z_\theta} \bigtriangledown_\theta Z_\theta \\ &= -\bigtriangledown_\theta U(E_i;\theta) + \mathbb{E}_{E \sim q_\theta}\left[\bigtriangledown_\theta U(E;\theta)\right] \end{aligned} \tag{8}$$

For parameter update in epoch t, we have the following rule to do gradient descent:

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \epsilon(\mathbb{E}_{E_i \sim p_D}\left[\bigtriangledown_\theta U(E_i;\theta)\right] - \mathbb{E}_{E_i \sim q_\theta}\left[\bigtriangledown_\theta U(E_i;\theta)\right]) \tag{9}$$

where ϵ is the learning step. If we take a closer look at the update equation, we can find there are two terms describing the gradient of energy function over different

sample distributions, i.e. p_D and q_θ, standing for the true distribution and the edge density estimator distribution respectively. Minimizing the difference of these two terms is equivalent to approximating q_θ to p_D by updating θ.

However, due to the sparsity of the edge probability space and uncertainty in the data modes, sampling from the edge density estimator using MCMC methods may take too much time. Hence, we employ a neural generator G_ϕ to approximate the model distribution p_θ.

One measure for the distance between two probability distributions is Kullback–Leibler divergence. Denote the distribution defined by the neural generator G_ϕ as q_ϕ, then the KL divergence between q_ϕ and q_θ can be expressed as:

$$\begin{aligned}\mathrm{KL}(\mathrm{q}_\phi \| \mathrm{q}_\theta) &= \sum_{E_i \in \mathcal{E}} q_\phi(E_i) \log \frac{q_\phi(E_i)}{q_\theta(E_i)} \\ &= -\mathrm{H}(E) + \mathbb{E}_{E_i \sim q_\phi}\left[U(E_i;\theta)\right] + \log Z_\theta \quad (10)\end{aligned}$$

The training objective of the neural generator G_ϕ is to minimize the KL divergence by updating ϕ. Under an ideal circumstance, p_ϕ and q_θ have the same distribution. Hence, we can substitute p_ϕ for q_θ in Eq. 9:

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \epsilon(\mathbb{E}_{E_i \sim p_D}\left[\bigtriangledown_\theta U(E_i;\theta)\right] - \mathbb{E}_{E_i \sim q_\phi}\left[\bigtriangledown_\theta U(E_i;\theta)\right]) \quad (11)$$

And the parameter ϕ of neural generator can be updated according to the rule:

$$\phi^{(t+1)} \leftarrow \phi^{(t)} - \gamma(-\bigtriangledown_\phi H(E) + \bigtriangledown_\phi \mathbb{E}_{E_i \sim q_\phi}\left[U(E_i;\theta)\right]) \quad (12)$$

where γ is the learning step. Notice that the third term in Eq. 10, i.e. the logarithm of the normalization term is a constant with respect to the model parameter ϕ. Its gradient is zero and thus omitted in the gradient update in Eq. 12.

D. Wasserstein Distance for Graph Construction

Eq. 11 shows that the optimization function is set as a kind of distance between the distribution of the energy model and the generator. Here we give a proof that the adopted measure is mathematically the Wasserstein distance. For two graphs generated from energy model and generator, we denote the probability space as $(\Omega_\theta, \mathcal{F}_\theta, \mathcal{P}_\theta) \rightarrow (\Xi, \mathcal{C})$ and $(\Omega_\phi, \mathcal{F}_\phi, \mathcal{P}_\phi) \rightarrow (\Xi, \mathcal{C})$ respectively. The Wasserstein distance between two distributions is written as:

$$W(\mathcal{P}_\theta, \mathcal{P}_\phi) = \inf_{\psi \in \Psi(\mathcal{P}_\theta, \mathcal{P}_\phi)} \mathbb{E}_{(\xi_\theta, \xi_\phi) \sim \psi}\left[\| \xi_\theta - \xi_\phi \|\right] \quad (13)$$

where $\Psi(\mathcal{P}_\theta, \mathcal{P}_\phi)$ denotes the set of all joint distributions $\psi(\xi_\theta, \xi_\phi)$ whose marginals are $\mathcal{P}_\theta$ and $\mathcal{P}_\phi$.

To complete the definition of Eq. 13, we need to further clarify the term characterizing the distance of two graphs $\| \xi_\theta - \xi_\phi \|$. For simplicity, here we only consider the circumstance where the two graphs have the same number of edges. Suppose $\xi_\theta = \{x_1, x_2, ..., x_n\}$ and $\xi_\phi = \{y_1, y_2, ..., y_n\}$. Then the two graphs are deemed as discrete distributions $\mu^\theta = \sum_{i=1}^n \frac{1}{n}\delta_{x_i}$ and $\mu^\phi = \sum_{i=1}^n \frac{1}{n}\delta_{y_i}$. We take $C_{ij} = \| x_i - y_j \|_*$, where $\|\cdot\|_*$ is the norm in $\mathcal{S}$. The optimal transport is a permutation of the index: $\|x_i - y_j\|_* = \min_\sigma \sum_{i=1}^n \|x_i - y_{\sigma(i)}\|_*$, where the minimum is taken among all $n!$ permutations.

Eq. 13 is computationally highly intractable and its dual form is usually utilized [19]:

$$W(\mathcal{P}_\theta, \mathcal{P}_\phi) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x \sim \mathcal{P}_\theta}\left[f(x)\right] - \mathbb{E}_{y \sim \mathcal{P}_\phi}\left[f(y)\right] \quad (14)$$

where f is called the Lipschitz functions. An alternative way widely used is to change the problem into an optimization problem:

$$\max_{\|f\|_L \leq 1} \mathbb{E}_{x \sim \mathcal{P}_\theta}\left[f(x)\right] - \mathbb{E}_{y \sim \mathcal{P}_\phi}\left[f(y)\right] \quad (15)$$

This is the Wasserstein distance of graph construction and it conforms with our optimization function denoted by Eq. 11. One insight the Wasserstein distance gives us is that the choice of Lipschitz function should obey the norm constraint. Hence, we add a regularization term called gradient penalty to Eq. 11. We update Eq. 11 by:

$$\begin{aligned}\theta^{(t+1)} \leftarrow \theta^{(t)} - \epsilon(&\mathbb{E}_{E_i \sim p_D}\left[\bigtriangledown_\theta U(E_i;\theta)\right] \quad (16)\\ &- \mathbb{E}_{E_i \sim q_\phi}\left[\bigtriangledown_\theta U(E_i;\theta)\right] + \lambda \mathbb{E}_{E_i \sim p_D}\left[\bigtriangledown_\theta \| \bigtriangledown_{E_i} U_\theta(E_i)\|^2\right])\end{aligned}$$

where λ is a weight for the gradient penalty term.

E. Neural Edge Sampling via Gumbel Softmax

To solve the intractability of the normalization constant, we adopt a neural generator to generate distribution approximate to that of the energy model. Here we specify the architecture and techniques used in the neural generator.

We use random walk process as the backbone of the neural generator to sample fake edges. At the beginning of random walk, we sample a random variable $n_1 \sim$ Uniform $(0, N_o)$, which works as the noise index for the starting point. In each walking step, the generator will output a normalized weight vector W_i for the current node i, i.e. $\sum_{n=1}^{N_o} w_{in} = 1$ where w_{ii} is set to zero manually to avoid self loop. A conventional way to pick the next index from the index pool is to sample from the categorical distribution parameterized by W_i. However, this sampling process is intrinsically non-differentiable, and hence is not appropriate to be incorporated in the gradient-based optimization methods. A common alternative approach is to use softmax and ensure the output vector is concentrated around the positions of the largest input values. Then the generator can select the index that has the maximum value as the next step's start. This approach is plagued with another problem. With a given input index, it will almost definitely outputs the same index. It deviates from the goal that we want to model the distribution of edge sampling and will ultimately lead to mode collapse.

Therefore, the design of edge sampling process should consider two main aspects: 1) a differentiable module, 2)

Algorithm 1 Training Process of GraphEBM

Input: Feature matrix $\mathcal{F}$, Partly Observed Graph $G_o = \{V_o, E_o\}$, Energy-based edge density estimator E_θ, Neural sampler G_ϕ.
Output: Model parameters θ, ϕ.

Initialize the model parameters $\theta^{(0)}$, $\phi^{(0)}$ randomly;
for training epoch t in 1 to T **do**
    // Energy model training phase.
    Sample true edges from G_o, $\{E_o^{(t)}\} \sim p_d(E)$;
    Sample fake edges from G_ϕ, $\{E_f^{(t)}\} \sim p_\phi(E)$ according to Alg. 2;
    Compute the energy value A_{ij} of each edge using Eq. 4, 5;
    Update $\theta^{(t)}$ according to Eq. 11;
    //Neural sampler training phase.
    Sample fake edges from G_ϕ, $\{E_f^{(t)}\} \sim p_\phi(E)$ according to Alg. 2;
    Update $\phi^{(t)}$ according to Eq. 12;
return θ^T, ϕ^T;

Algorithm 2 Neural Sampling Process

Input: Feature matrix $\mathcal{F}$, Neural sampler G_ϕ.
Output: Set of sampled fake edges $\{E_f\}$.

$E_f = \{\}$;
Sample the starting point index i_0 of random walk;
Lookup F to retrieve the feature embedding f_{i_0};
for random walk step s in 1 to S **do**
    Calculate the weight vector W_{i_s} from G_ϕ by: $W_{i_s} = G_\phi(f_{i_s})$;
    Sample an auxiliary variable $U \sim \mathrm{Uniform}(0, 1)$;
    Generate gumbel noise G according to Eq. 17;
    Calculate the noised weight vector W according to Eq. 18;
    Calculate a one-hot designation vector D_{i_s} according to Eq. 19;
    Append the selected E_{i_s} to E_f;
return E_f;

a stochastic process. Motivated by these requirement, we adopt gumbel-softmax reparameterization [20] to sample from the model-defined distribution. Consider the weight vector generated by the neural sampler is W_i, which is a $\mathbb{R}^{N_o}$ vector. And we sample a set of gumbel noise G from a standard Gumbel distribution. In practice, it can be generated by:

$$G = -\log(-\log(U)), \quad s.t. \;\; U \sim \mathrm{Uniform}(0, 1) \tag{17}$$

And the gumbel noises are added to the weight vector in an element-wise manner, i.e.

$$W_i^{'} = \left[w_{i1} + G^{(1)}, w_{i2} + G^{(2)}, ..., w_{iN_o} + G^{(N_o)}\right] \tag{18}$$

Then a softmax function is applied to the noised weight vector $W_i^{'}$ and gets a one-hot designation vector D for the next step:

$$D_{ij} = e^{(w_{ij}+G^{(j)})/\gamma} / \sum_{m=0}^{N_o} e^{(w_{im}+G^{(m)})/\gamma} \tag{19}$$

The next index can be retrieved by $j = \arg\max_j(D_{ij})$. In practice, we use the Straight Through Estimator (STE) [21] trick to retain the gradient for the training process. The gumbel softmax is intuitively effective here because it satisfies a rounding property, $P(W_{ik}^{'} > W_{il}^{'}, \; \forall \; k \neq l) = w_{ik}/w_{il}$.

F. Property Analysis

As we have discussed in Section I, our desired model should be accommodated to the scenario having frequent updates to the dataset. Here we explicate how the design in our model can meet these requirements and show these designs are non-trivial.

The existing works that are based on first learning an affinity matrix and then do node-wise sampling cannot accomplish this since they are intrinsically transductive models. When a new node arrives, the selection of neighborhood for each node is rerun, and thus the graph construction process takes at least an $O(n^2)$ complexity where n is the number of data points. By comparison, the energy model is learning the edge distribution and our graph construction procedure is based on sampling edges from an edge pool. First, the model can generalize to unseen data points and doesn't need to be retrained. And for the graph construction, we can easily come up with a hierarchical sampling algorithm for linear complexity by a quasi-dynamic programming approach. The idea is that we retain the edge samples from the original dataset, and maintain a numeric for the overall probability (a.k.a the normalization constant Z). When we have an unseen data point, we can easily derive the pairwise energy values by applying the pre-trained GraphEBM. The number of new possible edges involving this unseen data point is n. These n energy values also form a distribution where we can sample the newly-introduced edges. Here we generate a new graph by leveraging the previous sampling results and merely adding these new edges. Suppose the cumulative probability of the original graph is Z_1 and the sum of probabilities for the new edges are Z_2, then two edge pools are formed, one consisting of the previous edges, and the other consisting of the new edges. We only sample from Z_2 and the sampling time should correspond to a certain ratio to guarantee the relative number of previous and new edges. The whole sampling process can be seen as we first sample a binary variable, deciding which edge pool we are going to sample from. The ratio is decided by the cumulative probability values Z_1 and Z_2. This method is different from the per-node selection methods since it utilizes the previous

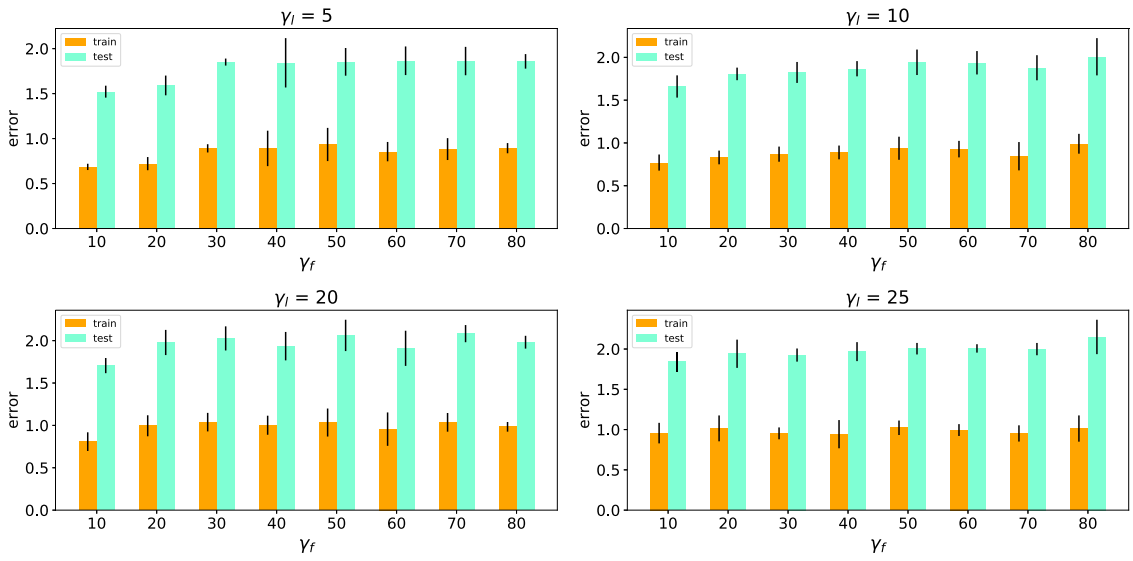


Figure 1. The mean absolute error change in semi-supervised learning with regard to the change in feature threshold γ_f, with different fixed label threshold $\gamma_l = \{5, 10, 20, 25\}$. Each bar plot shows the error on training set and testing set.

sampling results effectively. The update of dataset entails $O(n)$ complexity in graph reconstruction. Linear complexity algorithms for decrease in node number and modification of node features can also be trivially deduced using the above ideas.

IV. Experimental Results

In this section, we evaluate the performance of GraphEBM on a series of real-world datasets.

A. Dataset and Sample Graph Generation

The datasets we use are authentic public datasets collected from different domains, including 1) Student-Perf. [22], using student's information to predict student achievement in secondary education, especially in math subject; 2) Energy-Efc. [23], using different features of building shapes to predict heating load of the building; 3) Parkinsons. [24], using biomedical voice measurements from 42 people to predict early-stage Parkinson's disease; 4) Bike-Sharing. [25], using most of important events in the city to predict data on bike renting. This dataset is divided into two subtasks, one for predicting the activity of casual bike renting, denoted as Bike-Casual, and the other for predicting the registered users' bike renting, denoted by Bike-Registered. The statistics of these datasets are shown in Table. I,

Table I
Dataset Statistics

Methods	# of Instances	# of Attributes	Mean of Labels
Student-Perf.	649	33	10.899
Energy-Efc.	768	8	22.307
Parkinsons	1024	26	21.296
Bike-Casual	378	16	8.482
Bike-Registered	378	16	36.562

Generally, these datasets have numeric or categorical features and labels defined on a continuous domain. We try to find the underlying graph from these separate data points. One advantage of our model is that we can customize a graph as a handbook to provide a priori for GraphEBM to learn. The positive edges we feed into GraphEBM is generated from a sample graph. However, in most cases, the geometry of the structured data is often not revealed explicitly, especially when we only have access to individual data points. So first we adopt an empirical approach to construct the graph purely on the labeled points. To simulate the realistic scenarios, the generation of the sample graph should consider the following factors:

Limited Number of Labeled Data Points. In semi-supervised learning, the number of labeled points is highly limited. Here we follow this constraint. Denote N_l as the number of observable data points. The ratio of observable data points with regard to the whole dataset should be lower than 50%. The labeled data points are where we generate true edges to feed GraphEBM. They also provide the fixed labels in the inference stages, such as label propagation.

Label-level Smoothness. It is a consensual assumption for general-purpose graph-based semi-supervised learning that the closeness of nodes over a certain graph means the similarity of two data points in the label metric. Our work is built for such downstream applications and should follow this assumption. Hence we set a label-level threshold γ_l to restrict the choice of neighborhood. For node i, the logits

(unnormalized probability) of transition can be described as:

$$\mathbf{p}_i = \mathbf{1}_{|Y_i - Y_j| < \gamma_l} \odot (Y - Y_i) \tag{20}$$

where $\mathbf{1}_{[\cdot]}$ means a mask vector containing 1 at the entry where the condition is satisfied and 0 otherwise. $\odot$ is a Hadamard product operator between two vectors.

Feature-level Smoothness. Only guaranteeing the label-level smoothness in the recovered graph might incur redundant edges. Notice that the label of a node may be formulated by many factors in its features. For instance, a movie might be highly rated either because of its dramatic plots or due to the elegant costumes and stage settings. Such nodes with discrepant features and identical labels should not form an edge. We prune the $\mathbf{p}_i$ in Eq. 20 as follows:

$$\mathbf{p}_i = \mathbf{1}_{|X_i - X_j| < \gamma_f} \odot \mathbf{1}_{|Y_i - Y_j| < \gamma_l} \odot (Y - Y_i) \tag{21}$$

where γ_f is the predefined feature-level threshold.

B. Parameter Selection and Model Sensitivity

With the factors stated above, we have three hyper-parameters to decide how to generate the sample graph. Here we investigate how the difference of inputs can interfere with the model performance. The range of three parameters are listed as: $N_l \in \{128, 256\}$, $\gamma_l \in \{5, 10, 15, 20, 25\}$, $\gamma_f \in \{10\%, 20\%, 30\%, 40\%, 50\%, 60\%, 70\%, 80\%\}$. And we evaluate the model performance by mean absolute error between prediction and ground truth. The predicted value is derived from label propagation on the generated graph. The classic label propagation algorithm can be abbreviated as follows: we fix the labels of labeled data points, and update those unlabeled with the weighted average of its direct neighbors (labeled or unlabeled) till convergence for all the inferred labels. In this subsection, we conduct the following experiments on the Energy-Efc. dataset.

Fig. 1 shows the relation between the error and the feature threshold, with $N_l = 256$ and results under different γ_l are displayed separately. The left bar of each column means the error on training set and the right for testing set. We can observe that for almost any label threshold γ_l, the optimal choice of feature threshold should be as low as possible. And the error only changes slightly when γ_f is greater than 30%. It means that the edge selection for a node should be inclined to its direct neighborhood in the feature space. And the decrease in redundant edges is beneficial to the semi-supervised learning process. Also, it is observed that the improvement brought by lower feature threshold decays when the label threshold increases. This is because the worst case is not interfered greatly by the different settings.

Similarly, Fig. 2 shows the relation between the label threshold and the mean absolute errors. In each bar-plot, with the feature threshold fixed, we can observe that the performance is better when we feed a graph with lower label threshold. This effect is more significant for lower feature threshold. However, the above two experiments do not suggest that the feature threshold and label threshold should be as low as possible. They show a general trend for the error when modifying the threshold value. They also provide information on the importance of both guaranteeing the label-level smoothness and feature-level smoothness on graphs. In these experiments, we manage to generate a graph with almost all the nodes connected with a certain path. With the extreme settings, i.e. $\gamma_f = 0$ or $\gamma_l = 0$, the generated graph is deprecated to separate data points, which is not suitable for downstream applications on graph. The exact choice should be task-specific with regard to the intervals of data points in the feature and label space.

Fig. 3 shows the change of mean absolute error on the testing set with the number of labeled data points. We can observe that with more labeled data points used to generate the sample graph, the performance of our model can usually be slightly improved.

C. Performance on Semi-supervised Prediction Tasks

One of the ways to validate the effectiveness of our model is to deploy it in the downstream graph-based learning tasks and see if it can improve the performance by only replacing the input graph. We compare our GraphEBM with the following works:

- **kNN** (K Nearest Neighbours). kNN is the standard paradigm for current works on graph construction. This method calculates the distance for each node pair, and selects k nearest nodes in the feature space to form the edges for each node. In the experiment, we choose $k = \{3, 7\}$ for each node.
- **TPG** (Tensor Product Graph) [5]. Tensor Product Graph is representative for the physically motivated models for graph construction. It associates each node with a set of data points and do diffusion on this set to obtain more reliable similarities with higher-order information by the TPG.
- **RGCLI** [6]. RGCLI is an enhanced version of the classic GBILI [27] model. In this work, the label information is first utilized in the graph construction process. These methods take into account the position of the labeled points and integrate the evaluation on the distance with labeled points into neighborhood selection.
- **RGC** [26]. RGC (Robust Graph Construction) is proposed for dealing with noisy data. This method obeys the low-rank recovery and graph-smoothness assumption.

For the above methods, we generate graphs for different datasets, and feed them into a label propagation algorithm. We adopt a metric called normalized mean absolute error (NMAE). It calculates the relative deviation of the prediction results to the mean label values, so that this metric is comparable across datasets. The lower the metric is means

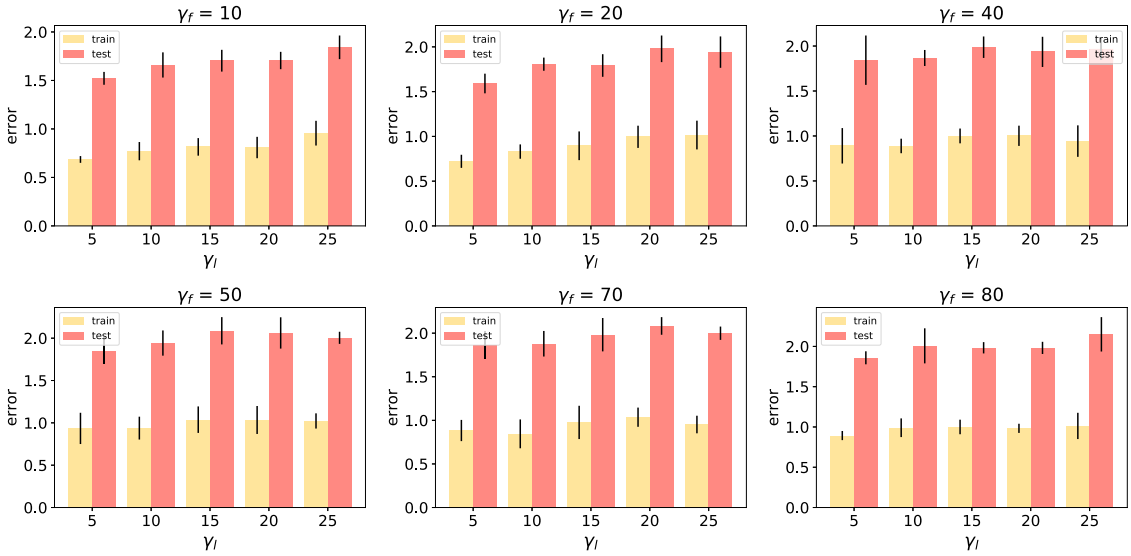


Figure 2. The mean absolute error change in semi-supervised learning with regard to the change in label threshold γ_l, with different fixed feature threshold $\gamma_f = \{10, 20, 40, 50, 70, 80\}$. Each bar plot shows the error on training set and testing set.

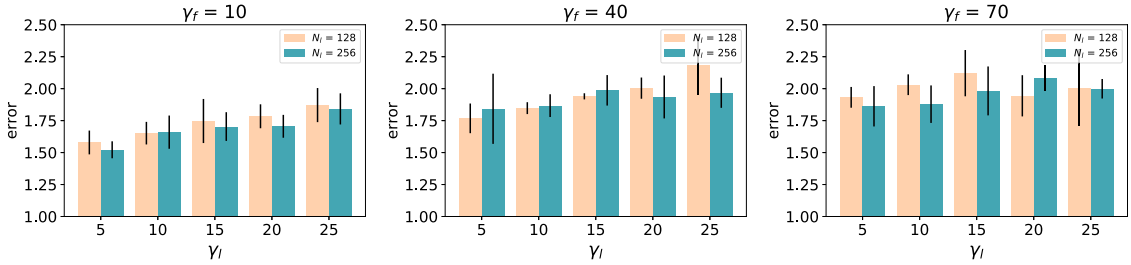


Figure 3. The comparison of the mean absolute error under different number of labeled data points $N_l \in \{128, 256\}$.

Table II
NORMALIZED MEAN ABSOLUTE ERROR (NMAE) OF SEMI-SUPERVISED PREDICTION OVER VARIOUS DATASETS (IN %).

Methods	Student-Perf. [22]	Energy-Efc. [23]	Parkinsons [24]	Bike-Casual [25]	Bike-Registered [25]
kNN(k=3)	23.80	10.22	23.01	34.23	24.67
kNN(k=7)	23.75	10.04	24.32	34.76	23.87
TPG [5]	23.62	8.63	22.85	35.66	21.22
RGCLI [6]	23.03	8.24	23.67	33.41	20.93
RGC [26]	23.20	8.55	22.63	34.47	22.68
GraphEBM(ours)	**22.18**	**6.81**	**20.78**	**29.39**	**16.83**

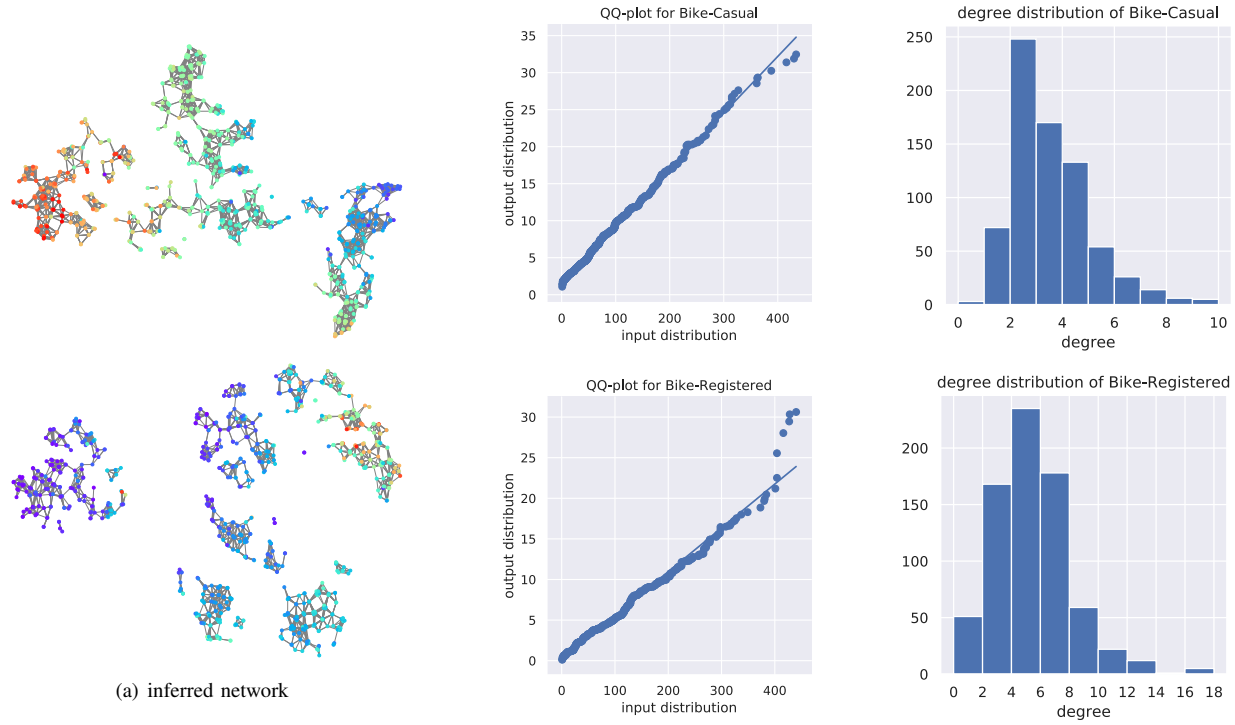
the better performance. NMAE is defined as below,

$$\text{NMAE} = \frac{1}{N}\sum_i |l_i - y_i| / \frac{1}{N}\sum_i y_i \tag{22}$$

where l_i and y_i are inferred and true labels respectively. Results are shown in Table. II. We can observe that our model consistently outperforms the baseline models. The improvement in NMAE ranges from 3.6% to 24.4%.

D. Learning Global Properties

Real-world graph datasets usually have latent global properties, which might be useful in representing the whole communities. However, current literature using greedy per-node neighborhood selection do not consider this rich information. With our design of Edge Probability Space and Wasserstein Learning, our proposed GraphEBM can to some extent learn the global property, although the guidance we provide is local pairwise connectivity. Here we choose two datasets on the same feature space but with different tasks, Bike-Casual and Bike-Registered, for analysis. The inferred graphs are shown in Fig. 4(a). We can observe some of the nodes are deemed as the peripheral nodes, since they have fewer connections. In kNN, when the hyper-parameter k is set, each node will select at least k neighbors, which is more likely to introduce noisy data points. This is one of the



(a) inferred network

(b) QQ-plot

(c) Degree Distribution

Figure 4. Global properties tested on two datasets, Bike-Casual (top) and Bike-Registered (bottom). Fig.a) shows the inferred graph underlying these two datasets. Fig.b) shows the QQ-plots. The x-axis is the learned edge distribution on the test set and the y-axis is the edge distribution in the sample graph. Each point indicates the quantile of two distributions. Fig.c) shows the degree distributions for two datasets.

reasons our model can outperform other models in Graph Semi-supervised Learning.

Although our guidance in supervised learning settings only consists of separate local connections of nodes, the deduction of Wasserstein distance in the objective function helps the model capture some graph-level, or the so-called global properties from the training set. Here we use Quantile-quantile plot (QQ-plot) to show the difference between edge distribution of the training set and the learned distribution. QQ-plot compares two distributions by plotting the quantiles against each other. The x-coordinate and y-coordinate of each point in the plot figure correspond to one specific quantile on two distributions. Fig. 4(b) shows the QQ-plot for the two datasets respectively. Since we do not care about the absolute values of energy but the relativeness of these values, the information of distribution can be preserved using QQ-plot. In Fig. 4(b), we observe that the points in the figure generally follow a certain line. It means the quantiles of the two distributions are increasing accordingly. The two distributions can agree after linearly transforming the values in either one of the distributions. The identity of two distributions endows our proposed GraphEBM with the capability of integrating more domain knowledge in learning the graph construction process. For some tasks with specific requirements on edge distribution, we can hand-craft a sample sub-graph with lower cost and let the model adaptively learn its useful edge information.

Other than the edge distribution, we also care about the degree distribution as a key indicator for the graph global property. In Fig. 4(c), we show the degree distribution for the two tasks. The two distributions are slightly different but share the same property that most of the nodes have few connections. This phenomenon can be compared with the uniform degree distribution by kNN. For different tasks on the same dataset, the graphs we learned are in different shapes, which can be seen as a verification for our task-specific learning purpose on the graph scale. Intuitively, the fixed edge distribution and flexible degree distribution are two regularizers for the whole graph, under which the model is learning the graph construction process, making the process both expressive and constrained. How the factors are interfering with the global graph construction is a very interesting problem. We leave it as the future work.

V. Conclusion

In this paper, we work on the graph construction problem, which is a fundamental and critical problem in graph-based

semi-supervised learning. We propose a probabilistic perspective to the problem, where the graph edges conform with a parametric distribution. The graph construction process is then converted to sampling from the Edge Probability Space. To capture the distribution, we propose a learning-based method called GraphEBM with the contrastive graph likelihood as the objective function. We also do comprehensive experiments to investigate different settings of the model and give guidance on choosing hyper-parameters. Comparing to the existing works on graph construction, our proposed GraphEBM can boost the performance of semi-supervised learning on various datasets. Also, we have shown that the model can capture global properties on the whole graph scale.

ACKNOWLEDGMENT

This material is based upon work supported by the National Science Foundation IIS 16-19302 and IIS 16-33755, Zhejiang University ZJU Research 083650, Futurewei Technologies HF2017060011 and 094013, UIUC OVCR CCIL Planning Grant 434S34, UIUC CSBS Small Grant 434C8U, and Advanced Digital Sciences Center Faculty Grant. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the funding agencies.

REFERENCES

[1] H. Bi, J. Sun, and Z. Xu, "A graph-based semisupervised deep learning model for polsar image classification," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 57, no. 4, pp. 2116–2132, 2018.

[2] Z. Qiu, E. Cho, X. Ma, and W. Campbell, "Graph-based semi-supervised learning for natural language understanding," in *Proceedings of the Thirteenth Workshop on Graph-Based Methods for Natural Language Processing (TextGraphs-13)*, 2019, pp. 151–158.

[3] A. Benamira, B. Devillers, E. Lesot, A. K. Ray, M. Saadi, and F. D. Malliaros, "Semi-supervised learning and graph neural networks for fake news detection," in *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, 2019, pp. 568–569.

[4] O. Chapelle, B. Scholkopf, and A. Zien, "Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]," *IEEE Transactions on Neural Networks*, vol. 20, no. 3, pp. 542–542, 2009.

[5] X. Yang, L. Prasad, and L. J. Latecki, "Affinity learning with diffusion on tensor product graph," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 1, pp. 28–38, 2012.

[6] L. Berton, T. de Paulo Faleiros, A. Valejo, J. Valverde-Rebaza, and A. de Andrade Lopes, "Rgcli: Robust graph that considers labeled instances for semi-supervised learning," *Neurocomputing*, vol. 226, pp. 238–248, 2017.

[7] K. Q. Weinberger, J. Blitzer, and L. K. Saul, "Distance metric learning for large margin nearest neighbor classification," in *Advances in neural information processing systems*, 2006, pp. 1473–1480.

[8] M. H. Rohban and H. R. Rabiee, "Supervised neighborhood graph construction for semi-supervised classification," *Pattern Recognition*, vol. 45, no. 4, pp. 1363–1372, 2012.

[9] Y. Du and I. Mordatch, "Implicit generation and modeling with energy based models," in *Advances in Neural Information Processing Systems*, 2019, pp. 3603–3613.

[10] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, "A learning algorithm for boltzmann machines," *Cognitive science*, vol. 9, no. 1, pp. 147–169, 1985.

[11] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural computation*, vol. 14, no. 8, pp. 1771–1800, 2002.

[12] R. Salakhutdinov and G. Hinton, "Deep boltzmann machines," in *Artificial intelligence and statistics*, 2009, pp. 448–455.

[13] A. Mnih and G. Hinton, "Learning nonlinear constraints with contrastive backpropagation," in *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, vol. 2. IEEE, 2005, pp. 1302–1307.

[14] G. Hinton, S. Osindero, M. Welling, and Y.-W. Teh, "Unsupervised discovery of nonlinear structure using contrastive backpropagation," *Cognitive science*, vol. 30, no. 4, pp. 725–731, 2006.

[15] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[16] J. Ngiam, Z. Chen, P. W. Koh, and A. Y. Ng, "Learning deep energy models," 2011.

[17] T. Kim and Y. Bengio, "Deep directed generative models with energy-based probability estimation," *arXiv preprint arXiv:1606.03439*, 2016.

[18] X. Dong, D. Thanou, M. Rabbat, and P. Frossard, "Learning graphs from data: A signal representation perspective," *IEEE Signal Processing Magazine*, vol. 36, no. 3, pp. 44–63, 2019.

[19] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," *arXiv preprint arXiv:1701.07875*, 2017.

[20] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," *arXiv preprint arXiv:1611.01144*, 2016.

[21] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in neural information processing systems*, 2015, pp. 3123–3131.

[22] P. Cortez and A. M. G. Silva, "Using data mining to predict secondary school student performance," 2008.

[23] A. Tsanas and A. Xifara, "Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools," *Energy and Buildings*, vol. 49, pp. 560–567, 2012.

[24] A. Tsanas, M. A. Little, P. E. McSharry, and L. O. Ramig, "Accurate telemonitoring of parkinson's disease progression by noninvasive speech tests," *IEEE transactions on Biomedical Engineering*, vol. 57, no. 4, pp. 884–893, 2009.

[25] H. Fanaee-T and J. Gama, "Event labeling combining ensemble detectors and background knowledge," *Progress in Artificial Intelligence*, vol. 2, no. 2-3, pp. 113–127, 2014.

[26] Z. Kang, H. Pan, S. C. Hoi, and Z. Xu, "Robust graph learning from noisy data," *IEEE transactions on cybernetics*, vol. 50, no. 5, pp. 1833–1843, 2019.

[27] L. Berton and A. de Andrade Lopes, "Graph construction based on labeled instances for semi-supervised learning," in *2014 22nd International Conference on Pattern Recognition*. IEEE, 2014, pp. 2477–2482.