# Discriminative Embeddings of Latent Variable Models for Structured Data

Hanjun Dai, Bo Dai, Le Song

College of Computing, Georgia Institute of Technology
{hanjundai, bodai}@gatech.edu, lsong@cc.gatech.edu

September 28, 2016

**Abstract**

Kernel classifiers and regressors designed for structured data, such as sequences, trees and graphs, have significantly advanced a number of interdisciplinary areas such as computational biology and drug design. Typically, kernels are designed beforehand for a data type which either exploit statistics of the structures or make use of probabilistic generative models, and then a discriminative classifier is learned based on the kernels via convex optimization. However, such an elegant two-stage approach also limited kernel methods from scaling up to millions of data points, and exploiting discriminative information to learn feature representations.

We propose, `structure2vec`, an effective and scalable approach for structured data representation based on the idea of embedding latent variable models into feature spaces, and learning such feature spaces using discriminative information. Interestingly, `structure2vec` extracts features by performing a sequence of function mappings in a way similar to graphical model inference procedures, such as mean field and belief propagation. In applications involving millions of data points, we showed that `structure2vec` runs 2 times faster, produces models which are $10,000$ times smaller, while at the same time achieving the state-of-the-art predictive performance.

## 1 Introduction

Structured data, such as sequences, trees and graphs, are prevalent in a number of interdisciplinary areas such as protein design, genomic sequence analysis, and drug design (Schölkopf et al., 2004). To learn from such complex data, we have to first transform such data explicitly or implicitly into some vectorial representations, and then apply machine learning algorithms in the resulting vector space. So far kernel methods have emerged as one of the most effective tools for dealing with structured data, and have achieved the state-of-the-art classification and regression results in many sequence (Leslie et al., 2002a; Vishwanathan & Smola, 2003) and graph datasets (Gärtner et al., 2003; Borgwardt, 2007).

The success of kernel methods on structured data relies crucially on the design of kernel functions — positive semidefinite similarity measures between pairs of data points (Schölkopf & Smola, 2002). By designing a kernel function, we have implicitly chosen a corresponding feature representation for each data point which can potentially has infinite dimensions. Later learning algorithms for various tasks and with potentially very different nature can then work exclusively on these pairwise kernel values without the need to access the original data points. Such modular structure of kernel methods has been very powerful, making them the most elegant and convenient methods to deal with structured data. Thus designing kernel for different structured objects, such as strings, trees and graphs, has always been an important subject in the kernel community. However, in the big data era, this modular framework has also limited kernel methods in terms of their ability to scale up to millions of data points, and exploit discriminative information to learn feature representations.

For instance, a class of kernels are designed based on the idea of "bag of structures" (BOS), where each structured data point is represented as a vector of counts for elementary structures. The spectrum kernel and variants for strings (Leslie et al., 2002a), subtree kernel (Ramon & Gärtner, 2003), graphlet kernel (Shervashidze et al., 2009) and Weisfeiler-lehman graph kernel (Shervashidze et al., 2011) all follow this design principle. In other words, the feature representations of these kernels are fixed before learning, with each dimension corresponding to a substructure, independent of the supervised learning tasks at hand. Since there are many unique substructures which may or may not be useful for the learning tasks, the explicit feature space of such kernels typically has very high dimensions. Subsequently algorithms dealing with the pairwise kernel values have to work with a big kernel matrix squared in the number of data points. The square dependency on the number of data points largely limits these BOS kernels to datasets of size just thousands.

A second class of kernels are based on the ingenious idea of exploiting the ability of probabilistic graphical models (GM) in describing noisy and structured data to design kernels. For instance, one can use hidden Markov models for sequence data, and use pairwise Markov random fields for graph data. The Fisher kernel (Jaakkola & Haussler, 1999) and probability product kernel (Jebara et al., 2004) are two representative instances within the family. The former method first fits a common generative model to the entire dataset, and then uses the empirical Fisher information matrix and the Fisher score of each data point to define the kernel; The latter method instead fits a different generative model for each data point, and then uses inner products between distributions to define the kernel. Typically the parameterization of these GM kernels are chosen before hand. Although the process of fitting generative models allow the kernels to adapt to the geometry of the input data, the resulting feature representations are still independent of the discriminative task at hand. Furthermore, the extra step of fitting generative models to data can be a challenging computation and estimation task by itself, especially in the presence of latent variables. Very often in practice, one finds that BOS kernels are easier to deploy than GM kernels, although the latter is supposed to capture the additional geometry and uncertainty information of data.

In this paper, we wish to revisit the idea of using graphical models for kernel or feature space design, with the goal of scaling up kernel methods for structured data to millions of data points, and allowing the kernel to learn the feature representation from label information. Our idea is to model each structured data point as a latent variable model, then embed the graphical model into feature spaces (Smola et al., 2007; Song et al., 2009), and use inner product in the embedding space to define kernels. Instead of fixing a feature or embedding space beforehand, we will also learn the feature space by directly minimizing the empirical loss defined by the label information.

The resulting embedding algorithm, `structure2vec`, runs in a scheme similar to graphical model inference procedures, such as mean field and belief propagation. Instead of performing probabilistic operations (such as sum, product and renormalization), the algorithm performs nonlinear function mappings in each step, inspired by kernel message passing algorithm in Song et al. (2010, 2011). Furthermore, `structure2vec` is also different from the kernel message passing algorithm in several aspects. First, `structure2vec` deals with a different scenario, *i.e.*, learning similarity measure for structured data. Second, `structure2vec` learns the nonlinear mappings using the discriminative information. And third, a variant of `structure2vec` can run in a mean field update fashion, different from message passing algorithms.

Besides the above novel aspects, `structure2vec` is also very scalable in terms of both memory and computation requirements. First, it uses a small and explicit feature map for the nonlinear feature space, and avoids the need for keeping the kernel matrix. This makes the subsequent classifiers or regressors order of magnitude smaller compared to other methods. Second, the nonlinear function mapping in `structure2vec` can be learned using stochastic gradient descent, allowing it to handle extremely large scale datasets.

Finally in experiments, we show that `structure2vec` compares favorably to other kernel methods in terms of classification accuracy in medium scale sequence and graph benchmark datasets including SCOP and NCI. Furthermore, `structure2vec` can handle extremely large data set, such as the 2.3 million molecule dataset from Harvard Clean Energy Project, run 2 times faster, produce model $10,000$ times smaller and achieve state-of-the-art accuracy. These strong empirical results suggest that the graphical models, theoretically well-grounded methods for capturing structure in data, combined with embedding techniques and discriminative training can significantly improve the performance in many large scale real-world structured data classification and regression problems.

## 2 Backgrounds

概率密度函数

We denote by $X$ a random variable with domain $\mathcal{X}$, and refer to instantiations of $X$ by the lower case character, $x$. We denote a density on $\mathcal{X}$ by $p(X)$, and denote the space of all such densities by $\mathcal{P}$. We will also deal with multiple random variables, $X_1, X_2, \ldots, X_\ell$, with joint density $p(X_1, X_2, \ldots, X_\ell)$. For simplicity of notation, we assume that the domains of all $X_t, t \in [\ell]$ are the same, but the methodology applies to the cases where they have different domains. In the case when $\mathcal{X}$ is a discrete domain, the density notation should be interpreted as probability, and integral should be interpreted as summation instead. Furthermore, we denote by $H$ a hidden variable with domain $\mathcal{H}$ and distribution $p(H)$. We use similar notation convention for variable $H$ and $X$.

**Kernel Methods.** Suppose the structured data is represented by $\chi \in \mathcal{G}$. Kernel methods owe the name to the use of kernel functions, $k(\chi, \chi') : \mathcal{G} \times \mathcal{G} \mapsto \mathbb{R}$, which are symmetric positive semidefinite (PSD), meaning that for all $n > 1$, and $\chi_1, \ldots, \chi_n \in \mathcal{G}$, and $c_1, \ldots, c_n \in \mathbb{R}$, we have $\sum_{i,j=1}^n c_i c_j k(\chi_i, \chi_j) \geqslant 0$. A signature of kernel methods is that learning algorithms for various tasks and with potentially very different nature can work exclusively on these pairwise kernel values without the need to access the original data points.

**Kernels for Structured Data.** Each kernel function will correspond to some feature map $\phi(\chi)$, where the kernel function can be expressed as the inner product between feature maps, *i.e.*, $k(\chi, \chi') = \langle \phi(\chi), \phi(\chi') \rangle$. For structured input domain, one can design kernels using counts on substructures. For instance, the spectrum kernel for two sequences $\chi$ and $\chi'$ is defined as (Leslie et al., 2002a)

$$k(\chi, \chi') = \sum\nolimits_{s \in \mathcal{S}} \#(s \in \chi) \#(s \in \chi') \tag{1}$$

where $\mathcal{S}$ is the set of possible subsequences, $\#(s \in x)$ counts the number occurrence of subsequence $s$ in $x$. In this case, the feature map $\phi(\chi) = (\#(s_1 \in \chi), \#(s_2 \in \chi), \ldots)^\top$ corresponds to a vector of dimension $|\mathcal{S}|$. Similarly, the graphlet kernel (Shervashidze et al., 2009) for two graphs $\chi$ and $\chi'$ can also be defined as (1), but $\mathcal{S}$ is now the set of possible subgraphs, and $\#(s \in \chi)$ counts the number occurrence of subgraphs. We refer to this class of kernels as "bag of structures" (BOS) kernel.

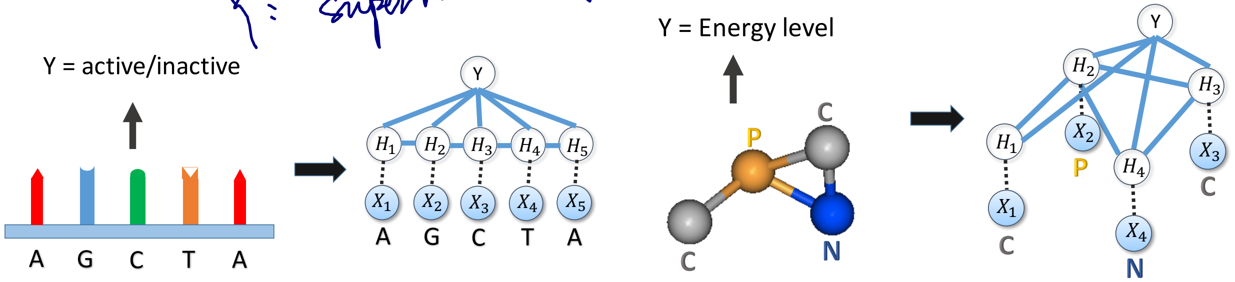
Kernels can also be defined by leveraging the power of probabilistic graphical models. For instance, the Fisher kernel (Jaakkola & Haussler, 1999) is defined using a parametric model $p(\chi|\theta^*)$ around its maximum likelihood estimate $\theta^*$, *i.e.*, $k(\chi, \chi') = U_\chi^\top I^{-1} U_{\chi'}$, where $U_\chi := \nabla_{\theta=\theta^*} \log p(\chi|\theta)$ and $I = \mathbb{E}_\mathcal{G}[U_\mathcal{G} U_\mathcal{G}^\top]$ is the Fisher information matrix. Another classical example along the line is the probability product kernel (Jebara et al., 2004). Different from the Fisher kernel based on generative model fitted with the whole dataset, the probability product kernel is calculated based on the models $p(\chi|\theta)$ fitted to individual data point, *i.e.*, $k(\chi, \chi') = \int_\mathcal{G} p(\tau|\theta_\chi)^\rho p(\tau|\theta_{\chi'})^\rho d\tau$ where $\theta_\chi$ and $\theta_{\chi'}$ are the maximum likelihood parameters for data point $\chi$ and $\chi'$ respectively. We refer to this class of kernels as the "graphical model" (GM) kernels.

**Hilbert Space Embedding of Distributions.** Hilbert space embeddings of distributions are mappings of distributions into potentially *infinite* dimensional feature spaces (Smola et al., 2007),

$$\mu_X := \mathbb{E}_X[\phi(X)] = \int_\mathcal{X} \phi(x) p(x) dx \ : \ \mathcal{P} \mapsto \mathcal{F} \qquad \mu_X \tag{2}$$

where the distribution is mapped to its expected feature map, *i.e.*, to a point in a feature space. Kernel embedding of distributions has rich representational power. Some feature map can make the mapping injective (Sriperumbudur et al., 2008), meaning that if two distributions, $p(X)$ and $q(X)$, are different, they are mapped to two distinct points in the feature space. For instance, when $\mathcal{X} = \mathbb{R}^d$, the feature spaces of many commonly used kernels, such as the Gaussian RBF kernel $\exp(-\|x - x'\|_2^2)$, can make the embedding injective.

Y: supervised information



(a) Represent string data as a latent variable model (b) Represent graph data as a latent variable model

Figure 1: Building graphical model with hidden variables from structured string and general graph data. $Y$ is the supervised information, which can be real number (for regression) or discrete integer (for classification).

单射 ↗ a point in the feature space

Alternatively, one can treat an injective embedding $\mu_X$ of a density $p(X)$ as a sufficient statistic of the density. Any information we need from the density is preserved in $\mu_X$: with $\mu_X$ one can uniquely recover $p(X)$, and any operation on $p(X)$ can be carried out via a corresponding operation on $\mu_X$ with the same result. For instance, this property will allow us to compute a functional $f : \mathcal{P} \mapsto \mathbb{R}$ of the density using the embedding only, *i.e.*,

$$f(p(x)) = \tilde{f}(\mu_X) \tag{3}$$

where $\tilde{f} : \mathcal{F} \mapsto \mathbb{R}$ is a corresponding function applied on $\mu_X$. Similarly the property can also be generalized to operators. For instance, applying an operator $\mathcal{T} : \mathcal{P} \mapsto \mathbb{R}^d$ to a density can also be equivalently carried out using its embedding, *i.e.*,

$$\mathcal{T} \circ p(x) = \tilde{\mathcal{T}} \circ \mu_X, \tag{4}$$

where $\tilde{\mathcal{T}} : \mathcal{F} \mapsto \mathbb{R}^d$ is the alternative operator working on the embedding. In our later sections, we will extensively exploit this property of injective embeddings, by assuming that there exists a feature space such that the embeddings are injective. We include the discussion of other related work in Appendix A.

## 3 Model for a Structured Data Point

为什么一定是图

Without loss of generality, we assume each structured data point $\chi$ is a graph, with a set of nodes $\mathcal{V} = \{1, \ldots, V\}$ and a set of edges $\mathcal{E}$. We will use $x_i$ to denote the value of the attribute for node $i$. We note the node attributes are different from the label of the entire data point. For instance, each atom in a molecule 分子 will correspond to a node in the graph, and the node attribute will be the atomic number, while the label for the entire molecule can be whether the molecule is a good drug or not. Other structures, such as sequences and trees, can be viewed as special cases of general graphs.

We will model the structured data point $\chi$ as an instance drawn from a graphical model. More specifically, we will model the label of each node in the graph with a variable $X_i$, and furthermore, associate an additional hidden variable $H_i$ with it. Then we will define a pairwise Markov random field on these collection of random variables

?

$$p(\{H_i\}, \{X_i\}) \propto \prod_{i \in \mathcal{V}} \Phi(H_i, X_i) \prod_{(i,j) \in \mathcal{E}} \Psi(H_i, H_j) \tag{5}$$

where $\Psi$ and $\Phi$ are nonnegative node and edge potentials respectively. In this model, the variables are connected according to the graph structure of the input data point. That is to say, we use the graph structure of the input data directly as the conditional independence structure of an undirected graphical model. Figure 1 illustrates two concrete examples in constructing the graphical models for strings and graphs. One can design more complicated graphical models which go beyond pairwise Markov random fields, and consider longer range interactions with potentials involving more variables. We will focus on pairwise Markov random fields for simplicity of representation.

marginal : 边际函数

posterior : 后验 条件概率

We note that such a graphical model is built for each individual data point, and the conditional independence structures of two graphical models can be different if the two data points $\chi$ and $\chi'$ are different. Furthermore, we do not observe the value for the hidden variables $\{H_i\}$, which makes the learning of the graphical model potentials $\Phi$ and $\Psi$ even more difficult. Thus, we will not pursue the standard route of maximum likelihood estimation, and rather we will consider the sequence of computations needed when we try to embed the posterior of $\{H_i\}$ into a feature space.

# 4 Embedding Latent Variable Models

We will embed the posterior marginal $p(H_i|\{x_i\})$ of a hidden variable using a feature map $\phi(H_i)$, *i.e.*,

$$\mu_i = \int_{\mathcal{H}} \phi(h_i) p(h_i|\{x_i\}) dh_i. \tag{6}$$

$\mu_i$ : embedding

The exact form of $\phi(H_i)$ and the parameters in MRF $p(H_i|\{x_i\})$ is not fixed at the moment, and we will learn them later using supervision signals for the ultimate discriminative target. For now, we will assume that $\phi(H_i) \in \mathbb{R}^d$ is a finite dimensional feature space, and the exact value of $d$ will determined by cross-validation in later experiments. However, compute the embedding is a very challenging task for general graphs: it involves performing an inference in graphical model where we need to integrate out all variables expect $H_i$, *i.e.*,

$$p(H_i|\{x_i\}) = \int_{\mathcal{H}^{\mathcal{V}-1}} p(H_i, \{h_j\}\,|\,\{x_j\}) \prod_{j\in\mathcal{V}\setminus i} dh_j. \tag{7}$$

Only when the graph structure is a tree, exact computation can be carried out efficiently via message passing (Pearl, 1988). Thus in the general case, approximate inference algorithms, *e.g.*, mean field inference and loopy belief propagation (BP), are developed. In many applications, however, these variational inference algorithms exhibit excellent empirical performance (Murphy et al., 1999). Several theoretical studies have also provided insight into the approximations made by loopy BP, partially justifying its application to graphs with cycles (Wainwright & Jordan, 2008; Yedidia et al., 2001a).

In the following subsection, we will explain the embedding of mean field and loopy BP. The embedding of other variational inference methods, *e.g.*, double-loop BP, damped BP, tree-reweighted BP, and generalized BP will be explained in Appendix D. We show that the iterative update steps in these algorithms, which are essentially minimizing approximations to the exact free energy, can be simply viewed as function mappings of the embedded marginals using the alternative view in (3) and (4).

## 4.1 Embedding Mean-Field Inference

The vanilla mean-field inference tries to approximate $p(\{H_i\}\,|\,\{x_i\})$ with a product of *independent* density components $p(\{H_i\}\,|\,\{x_i\}) \approx \prod_{i\in\mathcal{V}} q_i(h_i)$ where each $q_i(h_i) \geqslant 0$ is a valid density, such that $\int_{\mathcal{H}} q_i(h_i) dh_i = 1$. Furthermore, these density components are found by minimizing the following variational free energy (Wainwright & Jordan, 2008),

$$\min_{q_1,\ldots,q_d} \int_{\mathcal{H}^d} \prod_{i\in\mathcal{V}} q_i(h_i) \log \frac{\prod_{i\in\mathcal{V}} q_i(h_i)}{p(\{h_i\}\,|\,\{x_i\})} \prod_{i\in\mathcal{V}} dh_i.$$

One can show that the solution to the above optimization problem needs to satisfy the following fixed point equations for all $i \in \mathcal{V}$

$$\begin{aligned}
\log q_i(h_i) =& c_i + \log(\Phi(h_i, x_i)) + \sum_{j\in\mathcal{N}(i)} \int_{\mathcal{H}} q_j(h_j) \log(\Psi(h_i, h_j)\Phi(h_j, x_j)) dh_j \\
=& c_i' + \log \Phi(h_i, x_i) + \sum_{j\in\mathcal{N}(i)} \int_{\mathcal{H}} q_j(h_j) \log \Psi(h_i, h_j) dh_j
\end{aligned}$$

$W_1$ $W_2$

**Algorithm 1 Embedded Mean Field**

```
Input: parameter $\mathbf{W}$ in $\widetilde{\mathcal{T}}$
Initialize $\widetilde{\mu}_i^{(0)} = \mathbf{0}$, for all $i \in \mathcal{V}$
for $t = 1$ to $T$ do
    for $i \in \mathcal{V}$ do
        $l_i = \sum_{j\in\mathcal{N}(i)} \widetilde{\mu}_i^{(t-1)}$
        $\widetilde{\mu}_i^{(t)} = \sigma(W_1 x_i + W_2 l_i)$
    end for
end for{fixed point equation update}
return $\{\widetilde{\mu}_i^T\}_{i\in\mathcal{V}}$
```

$W_1, W_2, W_3, W_4$ Learn from data

**Algorithm 2 Embedding Loopy BP**

```
Input: parameter $\mathbf{W}$ in $\widetilde{\mathcal{T}}_1$ and $\widetilde{\mathcal{T}}_2$
Initialize $\widetilde{\nu}_{ij}^{(0)} = \mathbf{0}$, for all $(i,j) \in \mathcal{E}$
for $t = 1$ to $T$ do
    for $(i,j) \in \mathcal{E}$ do
        $\widetilde{\nu}_{ij}^t = \sigma(W_1 x_i + W_2 \sum_{k\in\mathcal{N}(i)\backslash j} \widetilde{\nu}_{ki}^{(t-1)})$
    end for
end for
for $i \in \mathcal{V}$ do
    $\widetilde{\mu}_i = \sigma(W_3 x_i + W_4 \sum_{k\in\mathcal{N}(i)\backslash j} \widetilde{\nu}_{ki}^{(T)})$
end for
return $\{\widetilde{\mu}_i\}_{i\in\mathcal{V}}$
```

where $c_i' = c_i + \sum_{j\in\mathcal{N}(i)} \int q_j(h_j) \log \Phi(h_j, x_j) dh_j$. Here $\mathcal{N}(i)$ are the set of neighbors of variable $H_i$ in the graphical model, and $c_i$ is a constant. The fixed point equations in (8) imply that $q_i(h_i)$ is a functional of a set of neighboring marginals $\{q_j\}_{j\in\mathcal{N}(i)}$, *i.e.*,

$$q_i(h_i) = f\left(h_i, x_i, \{q_j\}_{j\in\mathcal{N}(i)}\right). \tag{8}$$

If for each marginal $q_i$, we have an injective embedding

$$\widetilde{\mu}_i = \int_{\mathcal{H}} \phi(h_i) q_i(h_i) dh_i,$$

then, using similar reasoning as in (3), we can equivalently express the fixed point equation from an embedding point of view, *i.e.*, $q_i(h_i) = \tilde{f}(h_i, x_i, \{\widetilde{\mu}_j\}_{j\in\mathcal{N}(i)})$, and consequently using the operator view from (4), we have

$$\widetilde{\mu}_i = \widetilde{\mathcal{T}} \circ \left(x_i, \{\widetilde{\mu}_j\}_{j\in\mathcal{N}(i)}\right). \tag{9}$$

For the embedded mean field (9), the function $\tilde{f}$ and operator $\widetilde{\mathcal{T}}$ have complicated nonlinear dependencies on the potential functions $\Psi$, $\Phi$, and the feature mapping $\phi$ which is unknown and need to be learned from data. Instead of first learning the $\Psi$ and $\Phi$, and then working out $\widetilde{\mathcal{T}}$, we will pursue a different route where we directly parameterize $\widetilde{\mathcal{T}}$ and later learn it with supervision signals.

In terms of the parameterization, we will assume $\widetilde{\mu}_i \in \mathbb{R}^d$ where $d$ is a hyperparameter chosen using cross-validation. For $\widetilde{\mathcal{T}}$, one can use any nonlinear function mappings. For instance, we can parameterize it as a neural network

$$\widetilde{\mu}_i = \sigma\Big(W_1 x_i + W_2 \sum_{j\in\mathcal{N}(i)} \widetilde{\mu}_j\Big) \tag{10}$$

embedding learns from data

where $\sigma(\cdot) := \max\{0, \cdot\}$ is a rectified linear unit applied elementwisely to its argument, and $\mathbf{W} = \{W_1, W_2\}$. The number of the rows in $\mathbf{W}$ equals to $d$. With such parameterization, the mean field iterative update in the embedding space can be carried out as Algorithm 1. We could also multiply $\widetilde{\mu}_i$ with $V$ to rescale the range of message embeddings if needed. In fact, with or without $V$, the functions will be the same in terms of the representation power. Specifically, for any $(\mathbf{W}, V)$, we can always find another 'equivalent' parameters $(\mathbf{W}', I)$ where $\mathbf{W}' = \{W_1, W_2 V\}$.

## 4.2 Embedding Loopy Belief Propagation

Loopy belief propagation is another variational inference method, which essentially optimizes the Bethe free energy taking *pairwise* interactions into account (Yedidia et al., 2001b),

$$\min_{\{q_{ij}\}_{(i,j)\in\mathcal{E}}} -\sum_i (|\mathcal{N}(i)| - 1) \int_{\mathcal{H}} q_i(h_i) \log \frac{q_i(h_i)}{\Phi(h_i, x_i)} dh_i + \sum_{i,j} \int_{\mathcal{H}^2} q_{ij}(h_i, h_j) \log \frac{q_{ij}(h_i, h_j)}{\Psi(h_i, h_j)\Phi(h_i, x_i)\Phi(h_j, x_j)} dh_i dh_j$$

variational inference

subject to pairwise marginal consistency constraints: $\int_{\mathcal{H}} q_{ij}(h_i, h_j) dh_j = q_i(h_i)$, $\int_{\mathcal{H}} q_{ij}(h_i, h_j) dh_j = q_i(h_i)$, and $\int_{\mathcal{H}} q_i(h_i) dh_i = 1$. One can obtain the fixed point condition for the above optimization for all $(i, j) \in \mathcal{E}$,

$$m_{ij}(h_j) \propto \int_{\mathcal{H}} \prod_{k \in \mathcal{N}(i) \backslash j} m_{ki}(h_i) \Phi_i(h_i, x_i) \Psi_{ij}(h_i, h_j) dh_i,$$

$$q_i(h_i) \propto \Phi(h_i, x_i) \prod_{j \in \mathcal{N}(i)} m_{ji}(h_i). \tag{11}$$

where $m_{ij}(h_j)$ is the intermediate result called the message from node $i$ to $j$. Furthermore, $m_{ij}(h_j)$ is a nonnegative function which can be normalized to a density, and hence can also be embedded.

Similar to the reasoning in the mean field case, the (11) implies the messages $m_{ij}(h_j)$ and marginals $q_i(h_i)$ are functionals of messages from neighbors, *i.e.*,

$$m_{ij}(h_j) = f\left(h_j, x_i, \{m_{ki}\}_{k \in \mathcal{N}(i) \backslash j}\right),$$

$$q_i(h_i) = g\left(h_i, x_i, \{m_{ki}\}_{k \in \mathcal{N}(i)}\right).$$

With the assumption that there is an injective embedding for each message $\widetilde{\nu}_{ij} = \int \phi(h_j) m_{ij}(h_j) dh_j$ and for each marginal $\widetilde{\mu}_i = \int \phi(h_i) q_i(h_i) dh_i$, we can apply the reasoning from (3) and (4), and express the messages and marginals from the embedding view,

$$\widetilde{\nu}_{ij} = \widetilde{\mathcal{T}}_1 \circ \left(x_i, \{\widetilde{\nu}_{ki}\}_{k \in \mathcal{N}(i) \backslash j}\right), \tag{12}$$

$$\widetilde{\mu}_i = \widetilde{\mathcal{T}}_2 \circ \left(x_i, \{\widetilde{\nu}_{ki}\}_{k \in \mathcal{N}(i)}\right). \tag{13}$$

We will also use parametrization for loopy BP embedding similar to the mean field case, *i.e.*, neural network with rectified linear unit $\sigma$. Specifically, assume $\widetilde{\nu}_{ij} \in \mathbb{R}^d$, $\widetilde{\mu}_i \in \mathbb{R}^d$

$$\widetilde{\nu}_{ij} = \sigma\Big(W_1 x_i + W_2 \sum_{k \in \mathcal{N}(i) \backslash j} \widetilde{\nu}_{ki}\Big) \tag{14}$$

$$\widetilde{\mu}_i = \sigma\Big(W_3 x_i + W_4 \sum_{k \in \mathcal{N}(i)} \widetilde{\nu}_{ki}\Big) \tag{15}$$

where $\mathbf{W} = \{W_1, W_2, W_3, W_4\}$ are matrices with appropriate sizes. Note that one can use other nonlinear function mappings to parameterize $\widetilde{\mathcal{T}}_1$ and $\widetilde{\mathcal{T}}_2$ as well. Overall, the loopy BP embedding updates is summarized in Algorithm 2.

With similar strategy as in mean field case, we will learn the parameters in $\widetilde{\mathcal{T}}_1$ and $\widetilde{\mathcal{T}}_2$ later with supervision signals from the discriminative task.

### 4.3 Embedding Other Variational Inference

In fact, there are many other variational inference methods, with different forms of free energies or different optimization algorithms, resulting different message update forms, *e.g.*, double-loop BP (Yuille, 2002), damped BP (Minka, 2001), tree-reweightd BP (Wainwright et al., 2003), and generalized BP (Yedidia et al., 2001b). The proposed embedding method is a general technique which can be tailored to these algorithms. The major difference is the dependences in the messages. For the details of embedding of these algorithms, please refer to Appendix D.

## 5 Discriminative Training

Similar to kernel BP (Song et al., 2010, 2011) and kernel EP (Jitkrittum et al., 2015), our current work exploits feature space embedding to reformulate graphical model inference procedures. However, different from the kernel BP and kernel EP, in which the feature spaces are chosen beforehand and the conditional embedding operators are learned locally, our approach will learn both the feature spaces, the transformation $\widetilde{\mathcal{T}}$, as well as the regressor or classifier for the target values end-to-end using label information.

**Algorithm 3 Discriminative Embedding**

```
Input: Dataset $\mathcal{D} = \{\chi_n, y_n\}_{n=1}^N$, loss function $l(f(\chi), y)$.
Initialize $\mathbf{U}^0 = \{\mathbf{W}^0, \mathbf{u}^0\}$ randomly.
for $t = 1$ to $T$ do
  Sample $\{\chi_t, y_t\}$ uniform randomly from $\mathcal{D}$.
  Construct latent variable model $p(\{H_i^t\}|\chi_n)$ as (5).
  Embed $p(\{H_i^t\}|\chi_n)$ as $\{\widetilde{\mu}_i^n\}_{i\in\mathcal{V}_n}$ by Algorithm 1 or 2 with $\mathbf{W}^{t-1}$.
  Update $\mathbf{U}^t = \mathbf{U}^{t-1} + \lambda_t \nabla_{\mathbf{U}^{t-1}} l(f(\widetilde{\mu}^n; \mathbf{U}^{t-1}), y_n)$.
end for
return $\mathbf{U}^T = \{\mathbf{W}^T, \mathbf{u}^T\}$
```

graph instance?

Specifically, we are provided with a training dataset $\mathcal{D} = \{\chi_n, y_n\}_{n=1}^N$, where $\chi_n$ is a structured data point and $y_n \in \mathcal{Y}$, where $\mathcal{Y} = \mathbb{R}$ for regression or $\mathcal{Y} = \{1, \ldots, K\}$ for classification problem, respectively. With the feature embedding procedure introduced in Section 4, each data point will be represented as a set of embeddings $\{\tilde{\mu}_i^n\}_{i\in V_n} \in \mathcal{F}$. Now the goal is to learn a regression or classification function $f$ linking $\{\tilde{\mu}_i^n\}_{i\in V_n}$ to $y_n$.

More specifically, in the case of regression problem, we will parametrize function $f(\chi_n)$ as $u^\top \sigma(\sum_{i=1}^{V_n} \tilde{\mu}_i^n)$, where $u \in \mathbb{R}^d$ is the final mapping from summed (or pooled) embeddings to output. The parameters $u$ and those $\mathbf{W}$ involved in the embeddings are learned by minimizing the empirical square loss

$$\min_{u,\mathbf{W}} \sum_{n=1}^N \left( y_n - u^\top \sigma\left( \sum_{i=1}^{V_n} \widetilde{\mu}_i^n \right) \right)^2. \tag{16}$$

Note that each data point will have its own graphical model and embedded features due to its individual structure, but the parameters $u$ and $\mathbf{W}$, are shared across these graphical models.

In the case of $K$-class classification problem, we denote $z$ is the 1-of-$K$ representation of $y$, *i.e.*, $z \in \{0,1\}^K$, $z^k = 1$ if $y = k$, and $z^i = 0$, $\forall i \neq k$. By adopt the softmax loss, we obtain the optimization for embedding parameters and discriminative classifier estimation as,

$$\min_{\mathbf{u}=\{u^k\}_{k=1}^K, \mathbf{W}} \sum_n^N \sum_{k=1}^K -z_n^k \log u^k \sigma\left( \sum_{i=1}^{V_n} \widetilde{\mu}_i^n \right), \tag{17}$$

where $\mathbf{u} = \{u^k\}_{k=1}^K$, $u^k \in \mathbb{R}^d$ are the parameters for mapping embedding to output.

The same idea can also be generalized to other discriminative tasks with different loss functions. As we can see from the optimization problems (16) and (17), the objective functions are directly related to the corresponding discriminative tasks, and so as to $\mathbf{W}$ and $\mathbf{u}$. Conceptually, the procedure starts with representing each datum by a graphical model constructed corresponding to its *individual* structure with *sharing* potential functions, and then, we embed these graphical models with the *same* feature mappings. Finally the embedded marginals are aggregated with a prediction function for a discriminative task. The shared potential functions, feature mappings and final prediction functions are all learned together for the ultimate task with supervision signals.

We optimize the objective (16) or (17) with stochastic gradient descent for scalability consideration. However, other optimization algorithms are also applicable, and our method does not depend on this particular choice. The gradients of the parameters $\mathbf{W}$ are calculated recursively similar to recurrent neural network for sequence models. In our case, the recursive structure will correspond the message passing structure. The overall framework is illustrated in Algorithm 3. For details of the gradient calculation, please refer to Appendix E.

# 6 Experiments

Below we first compare our method with algorithms using prefixed kernel on string and graph benchmark datasets. Then we focus on Harvard Clean Energy Project dataset which contains 2.3 million samples. We demonstrate that while getting comparable performance on medium sized datasets, we are able to handle millions of samples, and getting much better when more training data are given. The two variants of `structure2vec` are denoted as DE-MF and DE-LBP, which stands for discriminative embedding using mean field or loopy belief propagation, respectively.

Our algorithms are implemented with C++ and CUDA, and experiments are carried out on clusters equipped with NVIDIA Tesla K20. The code is available on https://github.com/Hanjun-Dai/graphnn.

## 6.1 Benchmark structure datasets

We compare our algorithm on string benchmark datasets with the kernel method with existing sequence kernels, *i.e.*, the spectrum string kernel (Leslie et al., 2002a), mismatch string kernel (Leslie et al., 2002b) and fisher kernel with HMM generative models (Jaakkola & Haussler, 1999). On graph benchmark datasets, we compare with subtree kernel (Ramon & Gärtner, 2003) (R&G, for short), random walk kernel(Gärtner et al., 2003; Vishwanathan et al., 2010), shortest path kernel (Borgwardt & Kriegel, 2005), graphlet kernel(Shervashidze et al., 2009) and the family of Weisfeiler-Lehman kernels (WL kernel) (Shervashidze et al., 2011). After getting the kernel matrix, we train SVM classifier or regressor on top.

We tune all the methods via cross validation, and report the average performance. Specifically, for structured kernel methods, we tune the degree in $\{1, 2, \ldots, 10\}$ (for mismatch kernel, we also tune the maximum mismatch length in $\{1, 2, 3\}$) and train SVM classifier (Chang & Lin, 2001) on top, where the trade-off parameter $C$ is also chosen in $\{0.01, 0.1, 1, 10\}$ by cross validation. For fisher kernel that using HMM as generative model, we also tune the number of hidden states assigned to HMM in $\{2, \ldots, 20\}$.

For our methods, we simply use one-hot vector (the vector representation of discrete node attribute) as the embedding for observed nodes, and use a two-layer neural network for the embedding (prediction) of target value. The hidden layer size $b \in \{16, 32, 64\}$ of neural network, the embedding dimension $d \in \{16, 32, 64\}$ of hidden variables and the number of iterations $t \in \{1, 2, 3, 4\}$ are tuned via cross validation. We keep the number of parameters small, and use early stopping (Giles, 2001) to avoid overfitting in these small datasets.

### 6.1.1 String Dataset

Here we do experiments on two string binary classification benchmark datasets. The first one (denoted as SCOP) contains 7329 sequences obtained from SCOP (Structural Classification of Proteins) 1.59 database (Andreeva et al., 2004). Methods are evaluated on the ability to detect members of a target SCOP family (positive test set) belonging to the same SCOP superfamily as the positive training sequences, and no members of the target family are available during training. We use the same 54 target families and the same training/test splits as in remote homology detection (Kuang et al., 2005). The second one is FC and RES dataset (denoted as FC_RES) provided by CRISPR/Cas9 system, on which the task it to tell whether the guide RNA will direct Cas9 to target DNA. There are 5310 guides included in the dataset. Details of this dataset can be found in Doench et al. (2014); Fusi et al. (2015). We use two variants for spectrum string kernel: 1) kmer-single, where the constructed kernel matrix $K_k^{(s)}$ only consider patterns of length $k$; 2) kmer-concat, where kernel matrix $K^{(c)} = \sum_{i=1}^{k} K_k^{(s)}$. We also find the normalized kernel matrix $K_k^{Norm}(x, y) = \frac{K_k(x,y)}{\sqrt{K_k(x,x)K_k(y,y)}}$ helps.

| | FC_RES | SCOP |
|---|---|---|
| kmer-single | 0.7606±0.0187 | 0.7097±0.0504 |
| kmer-concat | 0.7576±0.0235 | 0.8467±0.0489 |
| mismatch | 0.7690±0.0197 | 0.8637±0.1192 |
| fisher | 0.7332±0.0314 | 0.8662±0.0879 |
| DE-MF | **0.7713±0.0208** | 0.9068±0.0685 |
| DE-LBP | 0.7701±0.0225 | **0.9167±0.0639** |

Table 1: Mean AUC on string classification datasets

Table 1 reports the mean AUC of different algorithms. We found two variants of `structure2vec` are consistently better than the string kernels. Also, the improvement in SCOP is more significant than in

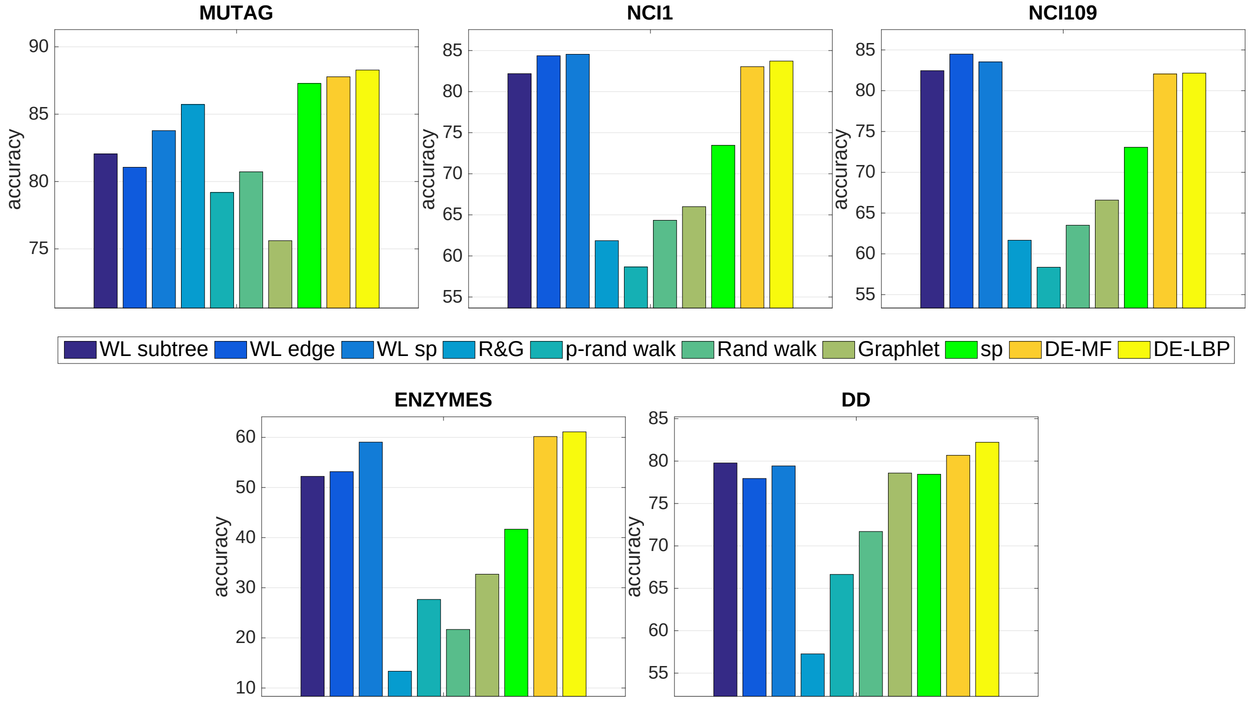


Figure 2: 10-fold cross-validation accuracies on graph classification benchmark datasets. The 'sp' in the figure stands for shortest-path.

| | size | avg $\|V\|$ | avg $\|E\|$ | #labels |
|---|---|---|---|---|
| MUTAG | 188 | 17.93 | 19.79 | 7 |
| NCI1 | 4110 | 29.87 | 32.3 | 37 |
| NCI109 | 4127 | 29.68 | 32.13 | 38 |
| ENZYMES | 600 | 32.63 | 62.14 | 3 |
| D&D | 1178 | 284.32 | 715.66 | 82 |

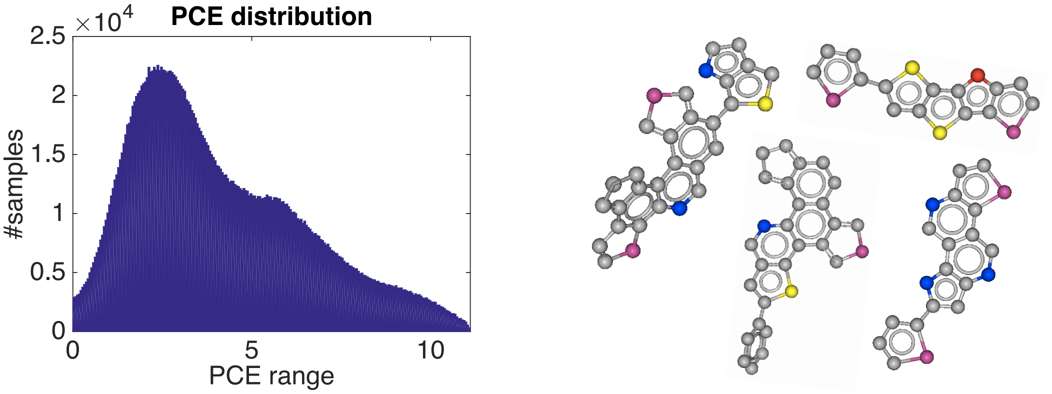
Table 2: Statistics (Sugiyama & Borgwardt, 2015) of graph benchmark datasets. $|V|$ is the # nodes while $|E|$ is the # edges in a graph. #labels equals to the number of different types of nodes.

FC_RES. This is because SCOP is a protein dataset and its alphabet size $|\Sigma|$ is much larger than that of FC_RES, an RNA dataset. Furthermore, the dimension of the explicit features for a k-mer kernel is $O(|\Sigma|^k)$, which can make the off-diagonal entries of kernel matrix very small (or even zero) with large alphabet size and $k$. That's also the reason why kmer-concat performs better than kmer-single. `structure2vec` learns a discriminative feature space, rather than prefix it beforehand, and hence does not have this problem.

### 6.1.2 Graph Dataset

We test the algorithms on five benchmark datasets for graph kernel: MUTAG, NCI1, NCI109, ENZYMES and D&D. MUTAG (Debnath et al., 1991). NCI1 and NCI109 (Wale et al., 2008) are chemical compounds dataset, while ENZYMES (Borgwardt & Kriegel, 2005) and D&D (Dobson & Doig, 2003) are of proteins. The task is to do multi-class or binary classification. We show the detailed statistics of these datasets in Table 2.

The results of baseline algorithms are taken from Shervashidze et al. (2011) since we use exactly the same setting here. From the accuracy comparison shown in Figure 2, we can see the proposed embedding methods are comparable to the alternative graph kernels, on different graphs with different number of labels,



(a) PCE distribution

(b) Sample molecules

Figure 3: PCE value distribution and sample molecules from CEP dataset. Hydrogens are not displayed.

nodes and edges. Also, in dataset D&D which consists of 82 different types of labels, our algorithm performs much better. As reported in Shervashidze et al. (2011), the time required for constructing dictionary for the graph kernel can take up to more than a year of CPU time in this dataset, while our algorithm can learn the discriminative embedding efficiently from structured data directly without the construction of the handcraft dictionary.

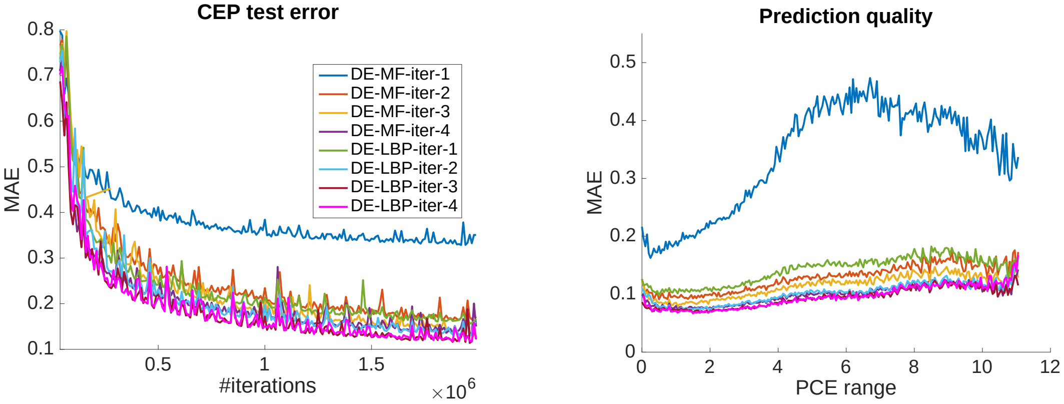
## 6.2 Harvard Clean Energy Project(CEP) dataset

The Harvard Clean Energy Project (Hachmann et al., 2011) is a theory-driven search for the next generation of organic solar cell materials. One of the most important properties of molecule for this task is the overall efficiency of the energy conversion process in a solar cell, which is determined by the power conversion efficiency (PCE). The Clean Energy Project (CEP) performed expensive simulations for the 2.3 million candidate molecules on IBMs World Community Grid, in order to get this property value. So using machine learning approach to accurately predict the PCE values is a promising direction for the high throughput screening and discovering new materials.

In this experiment, we randomly select 90% of the data for training, and the rest 10% for testing. This setting is similar to Pyzer-Knapp et al. (2015), except that we use the entire 2.3m dataset here. Since the data is distributed unevenly (see Figure 3), we resampled the training data (but not the test data) to make the algorithm put more emphasis on molecules with higher PCE values, in order to make accurate prediction for promising candidate molecules. Since the traditional kernel methods are not scalable, we make the explicit feature maps for WL subtree kernel by collecting all the molecules and creating dictionary for the feature space. The other graph kernels, like edge kernel and shortest path kernel, are having too large feature dictionary to work with. We use RDKit (Landrum, 2012) to extract features for atoms (nodes) and bonds (edges).

The mean absolute error (MAE) and root mean square error (RMSE) are reported in Table 3. We found utilizing graph information can accurately predict PCE values. Also, our proposed two methods are working equally well. Although WL tree kernel with degree 6 is also working well, it requires $10,000$ times more parameters than `structure2vec` and runs 2 times slower. The preprocessing needed for WL tree kernel also makes it difficult to use in large datasets.

To understand the effect of the inference embedding in the proposed algorithm framework, we further compare our methods with different number of fixed point iterations in Figure 4. It can see that, higher number of fixed point iterations will lead to faster convergence, though the number of parameters of the model in different settings are the same. The mean field embedding will get much worse result if only one iteration is executed. Compare to the loopy BP case with same setting, the latter one will always have one more round message passing since we need to aggregate the messages from edge to node in the last step.



(a) Test error vs iterations

(b) Prediction quality

Figure 4: Details of training and prediction results for DE-MF and DE-LBP with different number of fixed point iterations.

| | test MAE | test RMSE | # params |
|---|---|---|---|
| Mean Predictor | 1.9864 | 2.4062 | 1 |
| WL lv-3 | 0.1431 | 0.2040 | 1.6m |
| WL lv-6 | 0.0962 | 0.1367 | 1378m |
| DE-MF | 0.0914 | 0.1250 | 0.1m |
| DE-LBP | **0.0850** | **0.1174** | 0.1m |

Table 3: Test prediction performance on CEP dataset. WL lv-$k$ stands for Weisfeiler-lehman with degree $k$.

And also, from the quality of prediction we find that, though making slightly higher prediction error for molecules with high PCE values due to insufficient data, the variants of our algorithm are not overfitting the ‘easy’ (i.e., the most popular) range of PCE value.

## 7 Conclusion

We propose, `structure2vec`, an effective and scalable approach for structured data representation based on the idea of embedding latent variable models into feature spaces, and learning such feature spaces using discriminative information. Interestingly, `structure2vec` extracts features by performing a sequence of function mappings in a way similar to graphical model inference procedures, such as mean field and belief propagation. In applications involving millions of data points, we showed that `structure2vec` runs 2 times faster, produces models 10,000 times smaller, while at the same time achieving the state-of-the-art predictive performance. `structure2vec` provides a nice example for the general strategy of combining the strength of graphical models, Hilbert space embedding of distribution and deep learning approach, which we believe will become common in many other learning tasks.

**Acknowledgements.** This project was supported in part by NSF/NIH BIGDATA 1R01GM108341, ONR N00014-15-1-2340, NSF IIS-1218749, and NSF CAREER IIS-1350983.

## References

Andreeva, A., Howorth, D., Brenner, S. E., Hubbard, T. J., Chothia, C., and Murzin, A. G. Scop database in 2004: refinements integrate structure and sequence family data. *Nucleic acids research*, 32(suppl 1): D226–D229, 2004.

Borgwardt, K. M. *Graph Kernels*. PhD thesis, Ludwig-Maximilians-University, Munich, Germany, 2007.

Borgwardt, K. M. and Kriegel, H.-P. Shortest-path kernels on graphs. In *ICDM*, 2005.

Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

Chang, C. C. and Lin, C. J. *LIBSVM: a library for support vector machines*, 2001. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

Chen, L. C., Schwing, A. G., Yuille, A. L., and Urtasun, R. Learning deep structured models. *arXiv preprint arXiv:1407.2538*, 2014.

Debnath, A. K., Lopez de Compadre, R. L., Debnath, G., Shusterman, A. J., and Hansch, C. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *J Med Chem*, 34:786–797, 1991.

Dobson, P. D. and Doig, A. J. Distinguishing enzyme structures from non-enzymes without alignments. *J Mol Biol*, 330(4):771–783, Jul 2003.

Doench, J. G., Hartenian, E., Graham, D. B., Tothova, Z., Hegde, H., Smith, I., Sullender, M., Ebert, B. L., Xavier, R. J., and Root, D. E. Rational design of highly active sgrnas for crispr-cas9-mediated gene inactivation. *Nature biotechnology*, 32(12):1262–1267, 2014.

Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in Neural Information Processing Systems*, pp. 2215–2223, 2015.

Fusi, N., Smith, I., Doench, J., and Listgarten, J. In silico predictive modeling of crispr/cas9 guide efficiency. *bioRxiv*, 2015. doi: 10.1101/021568. URL http://biorxiv.org/content/early/2015/06/26/021568.

Gärtner, T., Flach, P.A., and Wrobel, S. On graph kernels: Hardness results and efficient alternatives. In Schölkopf, B. and Warmuth, M. K. (eds.), *Proceedings of Annual Conference. Computational Learning Theory*, pp. 129–143. Springer, 2003.

Caruana, R., Lawrence, S., and Giles, L. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In *Advances in Neural Information Processing Systems 13*, volume 13, pp. 402. MIT Press, 2001.

Hachmann, J., Olivares-Amaya, R., Atahan-Evrenk, S., Amador-Bedolla, C., Sánchez-Carrera, R. S., Gold-Parker, A., Vogt, L., Brockway, A. M., and Aspuru-Guzik, A. The harvard clean energy project: large-scale computational screening and design of organic photovoltaics on the world community grid. *The Journal of Physical Chemistry Letters*, 2(17):2241–2251, 2011.

Henaff, M., Bruna, J., and LeCun, Y. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.

Hershey, J. R., Roux, J. L., and Weninger, F. Deep unfolding: Model-based inspiration of novel deep architectures. *arXiv preprint arXiv:1409.2574*, 2014.

Heskes, T. Stable fixed points of loopy belief propagation are local minima of the bethe free energy. *Advances in Neural Information Processing Systems*, pp. 343–350. MIT Press, 2002.

Jaakkola, T. S. and Haussler, D. Exploiting generative models in discriminative classifiers. In Kearns, M. S., Solla, S. A., and Cohn, D. A. (eds.), *Advances in Neural Information Processing Systems 11*, pp. 487–493. MIT Press, 1999.

Jebara, T., Kondor, R., and Howard, A. Probability product kernels. *J. Mach. Learn. Res.*, 5:819–844, 2004.

Jitkrittum, W., Gretton, A., Heess, N., Eslami, S. M. A., Lakshminarayanan, B., Sejdinovic, D., and Szabó, Z. Kernel-based just-in-time learning for passing expectation propagation messages. In *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence, UAI 2015, July 12-16, 2015, Amsterdam, The Netherlands*, pp. 405–414, 2015.

Kuang, R., Ie, E., Wang, K., Wang, K., Siddiqi, M., Freund, Y., and Leslie, C. Profile-based string kernels for remote homology detection and motif extraction. *Journal of bioinformatics and computational biology*, 3(03):527–550, 2005.

Landrum, G. Rdkit: Open-source cheminformatics (2013), 2012.

Leslie, C., Eskin, E., and Noble, W. S. The spectrum kernel: A string kernel for SVM protein classification. In *Proceedings of the Pacific Symposium on Biocomputing*, pp. 564–575, Singapore, 2002a. World Scientific Publishing.

Leslie, C., Eskin, E., Weston, J., and Noble, W. S. Mismatch string kernels for SVM protein classification. In *Advances in Neural Information Processing Systems*, volume 15, Cambridge, MA, 2002b. MIT Press.

Li, Y., Tarlow, D., Brockschmidt, M., and Zemel, R.. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

Lin, G., Shen, C., Reid, I., and van den Hengel, A. Deeply learning the messages in message passing inference. In *Advances in Neural Information Processing Systems*, 2015.

Minka, T. The EP energy function and minimization schemes. *See www. stat. cmu. edu/minka/papers/learning. html, August*, 2001.

Mou, L., Li, G., Zhang, L., Wang, T., and Jin, Z.. Convolutional neural networks over tree structures for programming language processing. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

Murphy, K. P., Weiss, Y., and Jordan, M. I. Loopy belief propagation for approximate inference: An empirical study. In *UAI*, pp. 467–475, 1999.

Pearl, J. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufman, 1988.

Pyzer-Knapp, E. O., Li, K., and Aspuru-Guzik, A. Learning from the harvard clean energy project: The use of neural networks to accelerate materials discovery. *Advanced Functional Materials*, 25(41):6495–6502, 2015.

Ramon, J. and Gärtner, T. Expressivity versus efficiency of graph kernels. Technical report, First International Workshop on Mining Graphs, Trees and Sequences (held with ECML/PKDD'03), 2003.

Ross, S., Munoz, D., Hebert, M., and Bagnell, J. A. Learning message-passing inference machines for structured prediction. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2737–2744. IEEE, 2011.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

Schölkopf, B., Tsuda, K., and Vert, J.-P. *Kernel Methods in Computational Biology.* MIT Press, Cambridge, MA, 2004.

Schölkopf, B., and Smola, A. J. *Learning with Kernels.* MIT Press, Cambridge, MA, 2002.

Shervashidze, N., Vishwanathan, S. V. N., Petri, T., Mehlhorn, K., and Borgwardt, K. Efficient graphlet kernels for large graph comparison. *Proceedings of International Conference on Artificial Intelligence and Statistics*. Society for Artificial Intelligence and Statistics, 2009.

Shervashidze, N., Schweitzer, P., Van Leeuwen, E. J., Mehlhorn, K., and Borgwardt, K. M. Weisfeiler-lehman graph kernels. *The Journal of Machine Learning Research*, 12:2539–2561, 2011.

Smola, A. J., Gretton, A., Song, L., and Schölkopf, B. A Hilbert space embedding for distributions. In *Proceedings of the International Conference on Algorithmic Learning Theory*, volume 4754, pp. 13–31. Springer, 2007.

Song, L., Huang, J., Smola, A. J., and Fukumizu, K. Hilbert space embeddings of conditional distributions. In *Proceedings of the International Conference on Machine Learning*, 2009.

Song, L., Gretton, A., and Guestrin, C. Nonparametric tree graphical models. In *13th Workshop on Artificial Intelligence and Statistics*, volume 9 of *JMLR workshop and conference proceedings*, pp. 765–772, 2010.

Song, L., Gretton, A., Bickson, D., Low, Y., and Guestrin, C. Kernel belief propagation. In *Proc. Intl. Conference on Artificial Intelligence and Statistics*, volume 10 of *JMLR workshop and conference proceedings*, 2011.

Sriperumbudur, B., Gretton, A., Fukumizu, K., Lanckriet, G., and Schölkopf, B. Injective Hilbert space embeddings of probability measures. In *Proceedings of Annual Conference. Computational Learning Theory*, pp. 111–122, 2008.

Sugiyama, M. and Borgwardt, K. Halting in random walk kernels. In *Advances in Neural Information Processing Systems*, pp. 1630–1638, 2015.

Vishwanathan, S. V. N. and Smola, A. J. Fast kernels for string and tree matching. In Becker, S., Thrun, S., and Obermayer, K. (eds.), *Advances in Neural Information Processing Systems 15*, pp. 569–576. MIT Press, Cambridge, MA, 2003.

Vishwanathan, S. V. N., Schraudolph, N. N., Kondor, I. R., and Borgwardt, K. M. Graph kernels. *Journal of Machine Learning Research*, 2010. URL http://www.stat.purdue.edu/ vishy/papers/VisSchKonBor10.pdf. In press.

Wainwright, M., Jaakkola, T., and Willsky, A. Tree-reweighted belief propagation and approximate ML estimation by pseudo-moment matching. In *9th Workshop on Artificial Intelligence and Statistics*, 2003.

Wainwright, M. J. and Jordan, M. I. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, $1(1-2)$:1–305, 2008.

Wale, N., Watson, I. A., and Karypis, G. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems*, 14(3):347–375, 2008.

Yedidia, J. S., Freeman, W. T., and Weiss, Y. Generalized belief propagation. *Advances in Neural Information Processing Systems*, pp. 689–695. MIT Press, 2001a.

Yedidia, J.S., Freeman, W.T., and Weiss, Y. Bethe free energy, kikuchi approximations and belief propagation algorithms. Technical report, Mitsubishi Electric Research Laboratories, 2001b.

Yedidia, J.S., Freeman, W.T., and Weiss, Y. Constructing free-energy approximations and generalized belief propagation algorithms. *IEEE Transactions on Information Theory*, 51(7):2282–2312, 2005.

Yuille, A. L. Cccp algorithms to minimize the bethe and kikuchi free energies: Convergent alternatives to belief propagation. *Neural Computation*, 14:2002, 2002.

Zheng, S., Jayasumana, S., Romera-Paredes, B., Vineet, B., Su, Z., Du, D., Huang, C., and Torr, P. Conditional random fields as recurrent neural networks. *arXiv preprint arXiv:1502.03240*, 2015.

# Appendix

## A More Related Work

### A.1 Comparison with Neural Networks on Graphs

Neural network is also a powerful tool on graph structured data. Scarselli et al. (2009) proposed a neural network which generates features by solving a heuristic nonlinear system iteratively, and is learned using Almeida-Pineda algorithm. To guarantee the existence of the solution to the nonlinear system, there are extra requirements for the features generating function. From this perspective, the model in (Li et al., 2015) can be considered as an extension of (Scarselli et al., 2009) where the gated recurrent unit is used for feature generation. Rather than these heuristic models, our model is based on the principled graphical model embedding framework, which results in flexible embedding functions for generating features. Meanwhile, the model can be learned efficiently by traditional stochastic gradient descent.

There are several work transferring locality concept of convolutional neural networks (CNN) from Euclidean domain to graph case, using hierarchical clustering, graph Laplacian (Bruna et al., 2013), or graph Fourier transform (Henaff et al., 2015). These models are still restricted to problems with the same graph structure, which is not suitable for learning with molecules. Mou et al. (2016) proposed a convolution operation on trees, while the locality are defined based on parent-child relations. Duvenaud et al. (2015) used CNN to learn the circulant fingerprints for graphs from end to end. The dictionary of fingerprints are maintained using softmax of subtree feature representations, in order to obtain a differentiable model. If we unroll the steps in Algorithm 3, it can also be viewed as an end to end learning system. However, the structures of the proposed model are deeply rooted in graphical model embedding, from mean field and loopy BP, respectively. Also, since the parameters will be shared across different unrolling steps, we would have more compact model. As will be shown in the experiment section, our model is easy to train, while yielding good generalization ability.

### A.2 Comparison with Learning Message Estimator

By recognizing inference as computational expressions, inference machines (Ross et al., 2011) incorporate learning into the messages passing inference for CRFs. More recently, Hershey et al. (2014); Zheng et al. (2015); Lin et al. (2015) designed specific recurrent neural networks and convolutional neural networks for imitating the messages in CRFs. Although these methods share the similarity, *i.e.*, bypassing learning potential function, to the proposed framework, there are significant differences comparing to the proposed framework.

The most important difference lies in the learning setting. In these existing messages learning work (Hershey et al., 2014; Zheng et al., 2015; Lin et al., 2015; Chen et al., 2014), the learning task is still estimating the messages represented graphical models with designed function forms, *e.g.*, RNN or CNN, by maximizing loglikelihood. While in our work, we represented each structured data as a distribution, and the learning task is regression or classification over these distributions. Therefore, we treat the embedded models as samples, and learn the nonlinear mapping for embedding, and regressor or classifier, $f : \mathcal{P} \to \mathcal{Y}$, over these distributions jointly, with task-dependent user-specified loss functions.

Another difference is the way in constructing the messages forms, and thus, the neural networks architecture. In the existing work, the neural networks forms are constructed *strictly* follows the message updates forms (8) or (11). Due to such restriction, these works only focus on discrete variables with finite values, and is difficult to extend to continuous variables because of the integration. However, by exploiting the embedding point of view, we are able to build the messages with more *flexible* forms without losing the dependencies. Meanwhile, the difficulty in calculating integration for continuous variables is no longer a problem with the reasoning (3) and (4).

# B Derivation of the Fixed-Point Condition for Mean-Field Inference

In this section, we derive the fixed-point equation for mean-field inference in Section 4. As we introduced, the mean-field inference is indeed minimizing the variational free energy,

$$\min_{q_1,\ldots,q_d} L(\{q_i\}_{i=1}^d) := \int_{\mathcal{H}^d} \prod_{i\in\mathcal{V}} q_i(h_i) \log \frac{\prod_{i\in\mathcal{V}} q_i(h_i)}{p(\{h_i\} \mid \{x_i\})} \prod_{i\in\mathcal{V}} dh_i.$$

Plug the MRF (5) into objective, we have

$$\begin{aligned}
L(\{q_i\}_{i=1}^d) &= -\int_{\mathcal{H}^d} \prod_{i\in\mathcal{V}} q_i(h_i) \left( \log \Phi(h_i, x_i) + \sum_{j\in\mathcal{N}(i)} \log\left(\Psi(h_i,h_j)\Phi(h_j,x_j)\right) + \sum_{k\notin\mathcal{N}(i)} \log \left( \prod_{(k,l)\in\mathcal{E}} \Psi(h_k,h_l)\Phi(h_k,x_k) \right) \right) \prod_{i\in\mathcal{V}} dh_i \\
&+ \sum_{i\in\mathcal{V}} \int_{\mathcal{H}} q_i(h_i) \log q_i(h_i) dh_i \\
&= -\int q_i(h_i) \log \Phi(h_i,x_i) dh_i - \sum_{j\in\mathcal{N}(i)} \int q_i(h_i) \left( \int q_j(h_j) \log\left(\Psi(h_i,h_j)\Phi(h_j,x_j)\right) dh_j \right) dh_i + c_i \qquad (18) \\
&+ \sum_{i\in\mathcal{V}} \int_{\mathcal{H}} q_i(h_i) \log q_i(h_i) dh_i,
\end{aligned}$$

where $c_i = -\int \prod_{k\notin\mathcal{N}(i)} q_k(h_k) \left( \sum_{k\notin\mathcal{N}(i)} \log \left( \prod_{(k,l)\in\mathcal{E}} \Psi(h_k,h_l)\Phi(h_k,x_k) \right) \right) \prod_{k\notin\mathcal{N}(i)} dh_k$. Take functional derivatives of $L(\{q_i\}_{i=1}^d)$ w.r.t. $q_i(h_i)$, and set them to zeros, we obtain the fixed-point condition in Section 4,

$$\log q_i(h_i) = c_i + \log \Phi(h_i,x_i) + \sum_{j\in\mathcal{N}(i)} \int q_j(h_j) \log\left(\Psi(h_i,h_j)\Phi(h_j,x_j)\right) dh_j. \tag{19}$$

This fixed-point condition could be further reduced due to the independence between $h_i$ and $x_j$ given $h_j$, *i.e.*,

$$\begin{aligned}
\log q_i(h_i) &= c_i + \log\Phi(h_i,x_i) + \sum_{j\in\mathcal{N}(i)} \int q_j(h_j)\log\Psi(h_i,h_j)dh_j + \sum_{j\in\mathcal{N}(i)} \int q_j(h_j)\log\Phi(h_j,x_j)dh_j \qquad (20) \\
&= c_i' + \log\Phi(h_i,x_i) + \sum_{j\in\mathcal{N}(i)} \int q_j(h_j)\log\Psi(h_i,h_j)dh_j, \qquad (21)
\end{aligned}$$

where $c_i' = c_i + \sum_{j\in\mathcal{N}(i)} \int q_j(h_j)\log\Phi(h_j,x_j)dh_j$.

# C Derivation of the Fixed-Point Condition for Loopy BP

The derivation of the fixed-point condition for loopy BP can be found in Yedidia et al. (2001b). However, to keep the paper self-contained, we provide the details here. The objective of loopy BP is

$$\begin{aligned}
\min_{\{q_{ij}\}_{(i,j)\in\mathcal{E}}} \quad & -\sum_i (|\mathcal{N}(i)|-1) \int_{\mathcal{H}} q_i(h_i) \log \frac{q_i(h_i)}{\Phi(h_i,x_i)} dh_i + \sum_{i,j} \int_{\mathcal{H}^2} q_{ij}(h_i,h_j) \log \frac{q_{ij}(h_i,h_j)}{\Psi(h_i,h_j)\Phi(h_i,x_i)\Phi(h_j,x_j)} dh_i dh_j \\
\text{s.t.} \quad & \int_{\mathcal{H}} q_{ij}(h_i,h_j)dh_j = q_i(h_i), \quad \int_{\mathcal{H}} q_{ij}(h_i,h_j)dh_j = q_i(h_i), \quad \int_{\mathcal{H}} q_i(h_i)dh_i = 1.
\end{aligned}$$

Denote $\lambda_{ij}(h_j)$ is the multiplier to marginalization constraints $\int_{\mathcal{H}} q_{ij}(h_i,h_j)dh_i - q_j(h_j) = 0$, the Lagrangian is formed as

$$\begin{aligned}
L(\{q_{ij}\},\{q_i\},\{\lambda_{ij}\},\{\lambda_{ji}\}) \qquad &= -\sum_i (|\mathcal{N}(i)|-1)\int_{\mathcal{H}} q_i(h_i)\log\frac{q_i(h_i)}{\Phi(h_i,x_i)}dh_i \\
&+\sum_{i,j}\int_{\mathcal{H}^2} q_{ij}(h_i,h_j)\log\frac{q_{ij}(h_i,h_j)}{\Psi(h_i,h_j)\Phi(h_i,x_i)\Phi(h_j,x_j)}dh_idh_j \\
&-\sum_{i,j}\int_{\mathcal{H}}\lambda_{ij}(h_j)\Big(\int_{\mathcal{H}} q_{ij}(h_i,h_j)dh_i - q_j(h_j)\Big)dh_j \\
&-\sum_{i,j}\int_{\mathcal{H}}\lambda_{ji}(h_i)\Big(\int_{\mathcal{H}} q_{ij}(h_i,h_j)dh_j - q_i(h_i)\Big)dh_i
\end{aligned}$$

with normalization constraints $\int_{\mathcal{H}} q_i(h_i)dh_i = 1$. Take functional derivatives of $L(\{q_{ij}\},\{q_i\},\{\lambda_{ij}\},\{\lambda_{ji}\})$ with respect to $q_{ij}(h_i,h_j)$ and $q_i(h_i)$, and set them to zero, we have

$$\begin{aligned}
q_{ij}(h_i,h_j) &\propto \Psi(h_i,h_j)\Phi(h_i,x_i)\Phi(h_j,x_j)\exp(\lambda_{ij}(h_j)+\lambda_{ji}(h_i)), \\
q_i(h_i) &\propto \Phi(h_i,x_i)\exp\left(\frac{\sum_{k\in\mathcal{N}(i)}\lambda_{ki}(h_i)}{|\mathcal{N}(i)|-1}\right).
\end{aligned}$$

We set $m_{ij}(h_j) = \frac{q_j(h_j)}{\Phi(h_i,x_i)\exp(\lambda_{ij}(h_j))}$, therefore,

$$\prod_{k\in\mathcal{N}(i)} m_{ki}(h_i) \propto \exp\left(\frac{\sum_{k\in\mathcal{N}(i)}\lambda_{ki}(h_i)}{|\mathcal{N}(i)|-1}\right).$$

Plug it into $q_{ij}(h_i,h_j)$ and $q_i(h_i)$, we recover the loopy BP update for marginal belief and

$$\exp(\lambda_{ji}(h_i)) = \frac{q_i(h_i)}{\Phi(h_i,x_i)m_{ji}(h_i)} \propto \prod_{k\in N(i)\backslash j} m_{ki}(h_i).$$

The update rule for message $m_{ij}(h_j)$ can be recovered using the marginal consistency constraints,

$$\begin{aligned}
m_{ij}(h_j) &= \frac{q_j(h_j)}{\Phi(h_i,x_i)\exp(\lambda_{ij}(h_j))} = \frac{\int_{\mathcal{H}} q_{ij}(h_i,h_j)dh_i}{\Phi(h_i,x_i)\exp(\lambda_{ij}(h_j))} \\
&= \Phi(h_j,x_j)\exp(\lambda_{ij}(h_j))\frac{\int_{\mathcal{H}}\Psi(h_i,h_j)\Phi(h_i,x_i)\exp(\lambda_{ji}(h_i))dh_i}{\Phi(h_i,x_i)\exp(\lambda_{ij}(h_j))} \\
&\propto \int_{\mathcal{H}}\Psi(h_i,h_j)\Phi(h_i,x_i)\prod_{k\in N(i)\backslash j} m_{ki}(h_i)dh_i.
\end{aligned}$$

Moreover, we also obtain the other important relationship between $m_{ij}(h_j)$ and $\lambda_{ji}(h_i)$ by marginal consistency constraint and the definition of $m_{ij}(h_j)$,

$$m_{ij}(h_j) \propto \int \Psi(h_i,h_j)\Phi(h_j,x_j)\exp(\lambda_{ji}(h_i))dh_i.$$

# D Embedding Other Variational Inference

The proposed embedding is a general algorithm and can be tailored to other variational inference methods with message passing paradigm. In this section, we discuss the embedding for several alternatives, which optimize the primal and dual Bethe free energy, its convexified version and Kikuchi free energy, respectively. We will parametrize the messages with the same function class, *i.e.*, neural networks with ReLU. More generally, we can also treat the weights as parameters and learn them together.

## D.1 Double-Loop BP

Noticed the Bethe free energy can be decomposed into the summation of a convex function and a concave function, Yuille (2002) utilizes CCCP to minimize the Bethe free energy, resulting the double-loop algorithm. Take the gradient of Lagrangian of the objective function, and set to zero, the primal variable can be represented in dual form,

$$q_{ij}(h_i,h_j) \propto \Psi(h_i,h_j)\Phi(h_i,x_i)\Phi(h_j,x_j)\exp(\lambda_{ij}(h_j)+\lambda_{ji}(h_i)),$$
$$q_i(h_i) \propto \Phi(h_i,x_i)\exp\Big(|\mathcal{N}(i)|\gamma_s(h_i) - \sum_{k\in\mathcal{N}(i)}\lambda_{ki}(h_i)\Big),$$

The algorithm updates $\gamma$ and $\lambda$ alternatively,

$$\begin{aligned} 2\lambda_{ij}^{new}(h_j) &= |\mathcal{N}(j)|\gamma_i(h_i) - \sum_{k\in\mathcal{N}(j)\backslash i}\lambda_{kj}(h_j) - \log\int_{\mathcal{H}}\Psi(h_i,h_j)\Phi(h_i,x_i)\lambda_{ji}(h_i)dh_i,\\ \gamma_i^{new}(h_i) &= |\mathcal{N}(i)|\gamma_i(h_i) - \sum_{k\in\mathcal{N}(i)}\lambda_{ki}(h_i). \end{aligned}$$

Consider the $\lambda_{ij}$ as messages, with the injective embedding assumptions for corresponding messages, follow the same notation, we can express the messages in embedding view

$$\begin{aligned} \widetilde{\nu}_{ij} &= \widetilde{\mathcal{T}}_1\circ\Big(x_i,\{\widetilde{\nu}_{ki}\}_{k\in\mathcal{N}(i)\backslash j},\widetilde{\nu}_{ji},\widetilde{\mu}_i\Big), \quad \text{or} \quad \widetilde{\nu}_{ij} = \widetilde{\mathcal{T}}_1\circ\Big(x_i,\{\widetilde{\nu}_{ki}\}_{k\in\mathcal{N}(i)},\widetilde{\mu}_i\Big),\\ \widetilde{\mu}_i &= \widetilde{\mathcal{T}}_2\circ\Big(x_i,\{\widetilde{\nu}_{ki}\}_{k\in\mathcal{N}(i)},\widetilde{\mu}_i\Big). \end{aligned}$$

Therefore, we have the parametrization form as

$$\begin{aligned} \widetilde{\nu}_{ij} &= \sigma\left(W_1x_i + W_2\sum_{k\in\mathcal{N}(i)\backslash j}\widetilde{\nu}_{ki} + W_3\widetilde{\nu}_{ji} + W_4\widetilde{\mu}_i\right), \quad \text{or} \quad \widetilde{\nu}_{ij} = \sigma\left(W_1x_i + W_2\sum_{k\in\mathcal{N}(i)}\widetilde{\nu}_{ki} + W_3\widetilde{\mu}_i\right),\\ \widetilde{\mu}_i &= \sigma\left(W_5x_i + W_6\sum_{k\in\mathcal{N}(i)}\widetilde{\nu}_{ki} + W_7\widetilde{\mu}_i\right). \end{aligned}$$

where $\widetilde{\mu}_i\in\mathbb{R}^d$, $\widetilde{\nu}_{ij}\in\mathbb{R}^d$, and $\mathbf{W}=\{W_i\}$ are matrices with appropriate size.

## D.2 Damped BP

Instead of the primal form of Bethe free energy, Minka (2001) investigates the duality of the optimization,

$$\begin{aligned} \min_{\gamma}\max_{\lambda} \quad & \sum_i\Big(|\mathcal{N}(i)|-1\Big)\log\int_{\mathcal{H}}\Phi(h_i,x_i)\exp(\gamma_i(h_i))dh_i\\ & - \sum_{(i,j)\in\mathcal{E}}\log\int_{\mathcal{H}^2}\Psi(h_i,h_j)\Phi(h_i,x_i)\Phi(h_j,x_j)\exp(\lambda_{ij}(h_j)+\lambda_{ji}(h_i))dh_jdh_i, \end{aligned}$$

subject to $\Big(|\mathcal{N}(i)|-1\Big)\gamma_i(h_i)=\sum_{k\in\mathcal{N}(i)}\lambda_{ki}(h_i)$. Define message as

$$m_{ij}(h_j)\propto\int_{\mathcal{H}}\Psi(h_i,h_j)\Phi(h_j,x_j)\exp(\lambda_{ji}(h_i))dh_i,$$

the messages updates are

$$\begin{aligned} m_{ij}(h_i) &\propto \int_{\mathcal{H}}\Phi_i(h_i,x_i)\Psi_{ij}(h_i,h_j)\exp\Big(\frac{|\mathcal{N}(i)|-1}{|\mathcal{N}(i)|}\gamma_i(h_i)\Big)\frac{\prod_{k\in\mathcal{N}(i)}m_{ki}^{\frac{1}{|\mathcal{N}(i)|}}(h_i)}{m_{ji}(h_i)}dh_i,\\ \gamma_i^{new}(h_i) &\propto \frac{|\mathcal{N}(i)|-1}{|\mathcal{N}(i)|}\gamma_i(h_i) + \sum_{k\in\mathcal{N}(i)}\frac{1}{|\mathcal{N}(i)|}\log m_{ki}(h_i). \end{aligned}$$

and the

$$q(x_i) \propto \Phi(h_i, x_i) \exp\left( \frac{|\mathcal{N}(i)| - 1}{|\mathcal{N}(i)|} \gamma_i(h_i) \right) \prod_{k \in \mathcal{N}(i)} m_{ki}^{\frac{1}{|\mathcal{N}(i)|}}$$

which results the embedding follows the same injective assumption and notations,

$$\begin{aligned} \widetilde{\nu}_{ij} &= \widetilde{\mathcal{T}}_1 \circ \left( x_i, \{\widetilde{\nu}_{ki}\}_{k \in \mathcal{N}(i)}, \widetilde{\nu}_{ji}, \widetilde{\mu}_i \right), \\ \widetilde{\mu}_i &= \widetilde{\mathcal{T}}_2 \circ \left( x_i, \widetilde{\mu}_i, \{\widetilde{\nu}_{ki}\}_{k \in \mathcal{N}(i)} \right). \end{aligned}$$

and the parametrization,

$$\begin{aligned} \widetilde{\nu}_{ij} &= \sigma\left( W_1 x_i + W_2 \sum_{k \in \mathcal{N}(i)} \widetilde{\nu}_{ki} + W_3 \widetilde{\nu}_{ji} + W_4 \widetilde{\mu}_i \right), \\ \widetilde{\mu}_i &= \sigma\left( W_5 x_i + W_6 \widetilde{\mu}_i + W_7 \sum_{k \in \mathcal{N}(i)} \widetilde{\nu}_{ki} \right). \end{aligned}$$

It is interesting that after parametrization, the embeddings of double-loop BP and damped BP are essentially the same, which reveal the connection between double-loop BP and damped BP.

## D.3 Tree-reweighted BP

Different from loopy BP and its variants which optimizing the Bethe free energy, the tree-reweighted BP (Wainwright et al., 2003) is optimizing a convexified Bethe energy,

$$\begin{aligned} \min_{\{q_{ij}\}_{(i,j \in \mathcal{E})}} L = & \sum_i \int_{\mathcal{H}} q(h_i) \log q(h_i) dh_i + \sum_{i,j} v_{ij} \int_{\mathcal{H}^2} q_{ij}(h_i, h_j) \log \frac{q_{ij}(h_i, h_j)}{q_i(h_i) q_j(h_j)} dh_i dh_j \\ & - \sum_i \int_{\mathcal{H}} q(h_i) \log \Phi(h_i, x_i) dh_i - \sum_{i,j} \int_{\mathcal{H}^2} q_{ij}(h_i, h_j) \log \Psi(h_i, h_j) dh_i dh_j \end{aligned}$$

subject to pairwise marginal consistency constraints: $\int_{\mathcal{H}} q_{ij}(h_i, h_j) dh_j = q_i(h_i)$, $\int_{\mathcal{H}} q_{ij}(h_i, h_j) dh_j = q_i(h_i)$, and $\int_{\mathcal{H}} q_i(h_i) dh_i = 1$. The $\{v_{ij}\}_{(i,j) \in \mathcal{E}}$ represents the probabilities that each edge appears in a spanning tree randomly chose from all spanning tree from $G = \{\mathcal{V}, \mathcal{E}\}$ under some measure. Follow the same strategy as loopy BP update derivations, *i.e.*, take derivatives of the corresponding Lagrangian with respect to $q_i$ and $q_{ij}$ and set to zero, meanwhile, incorporate with the marginal consistency, we can arrive the messages updates,

$$\begin{aligned} m_{ij}(h_j) &\propto \int_{\mathcal{H}} \Psi_{ij}^{\frac{1}{v_{ji}}}(h_i, h_j) \Phi_i(h_i, x_i) \frac{\prod_{k \in \mathcal{H}(i) \setminus j} m_{ki}^{v_{ki}}(h_i)}{m_{ji}^{1 - v_{ij}}(h_i)} dh_i, \\ q_i(h_i) &\propto \Phi_i(h_i, x_i) \prod_{k \in \mathcal{N}(i)} m_{ki}^{v_{ki}}(h_i), \\ q_{ij}(h_i, h_j) &\propto \Psi_{ij}^{\frac{1}{v_{ji}}}(h_i, h_j) \Phi_i(h_i, x_i) \Phi_j(h_j, x_j) \frac{\prod_{k \in \mathcal{N}(i) \setminus j} m_{ki}^{v_{ki}}(h_i)}{m_{ji}^{1 - v_{ij}}(h_i)} \frac{\prod_{k \in \mathcal{N}(j) \setminus i} m_{kj}^{v_{kj}}(h_j)}{m_{ij}^{1 - v_{ji}}(h_j)}. \end{aligned}$$

Similarly, the embedded messages and the marginals on nodes can be obtained as

$$\begin{aligned} \widetilde{\nu}_{ij} &= \widetilde{\mathcal{T}}_1 \circ \left( x_i, \{\widetilde{\nu}_{ki}\}_{k \in \mathcal{N}(i) \setminus j}, \widetilde{\nu}_{ji}, \{v_{ki}\}_{k \in \mathcal{N} i \setminus j}, v_{ij} \right), \\ \widetilde{\mu}_i &= \widetilde{\mathcal{T}}_2 \circ \left( x_i, \{\widetilde{\nu}_{ki}, v_{ki}\}_{k \in \mathcal{N}(i)} \right). \end{aligned}$$

Parametrize these message in the same way, we obtain,

$$\begin{aligned}\widetilde{\nu}_{ij} &= \sigma\left(W_1 x_i + W_2 \sum_{k\in\mathcal{N}(i)\backslash j} \widetilde{v}_{ki}\nu_{ki} + W_3 \widetilde{v}_{ij}\nu_{ji}\right),\\ \widetilde{\mu}_{i} &= \sigma\left(W_4 x_i + W_5 \sum_{k\in\mathcal{N}(i)} \widetilde{v}_{ki}\nu_{ki}\right).\end{aligned}$$

Notice the tree-weighted BP contains extra parameters $\{v_{ij}\}_{(i,j)\in\mathcal{E}}$ which is in the spanning tree polytope as (Wainwright et al., 2003).

## D.4 Generalized Belief Propagation

The Kikuchi free energy is the generalization of the Bethe free energy by involving *high-order* interactions. More specifically, given the MRFs, we denote $R$ to be a set of regions, *i.e.*, some basic clusters of nodes, their intersections, the intersections of the intersections, and so on. We denote the $sub(r)$ or $sup(r)$, *i.e.*, subregions or superregions of $r$, as the set of regions completely contained in $r$ or containing $r$, respectively. Let $h_r$ be the state of the nodes in region $r$, then, the Kikuchi free energy is

$$\sum_{r\in R} c_r \left(\int q(h_r)\log\frac{q(h_r)}{\prod_{i,j\in r}\Psi(h_i,h_j)\prod_{i\in r}\Phi(h_i,x_i)}\right),$$

where $c_r$ is over-counting number of region $r$, defined by $c_r := 1 - \sum_{s\in sup(r)} c_s$ with $c_r = 1$ if $r$ is the largest region in $R$. It is straightforward to verify that the Bethe free energy is a special case of the Kikuchi free energy by setting the basic cluster as pair of nodes. The generalized loopy BP (Yedidia et al., 2001b) is trying to seek the stationary points of the Kikuchi free energy under regional marginal consistency constraints and density validation constraints by following messages updates,

$$\begin{aligned}m_{r,s}(h_s) &\propto \frac{\int \overline{\Psi}(h_r, x_{r\backslash s})\bar{m}_{r\backslash s}(h_{r\backslash s})dh_{r\backslash s}}{\bar{m}_{r,s}(h_s)},\\ q_r(h_r) &\propto \prod_{i\in r}\Phi(h_i,x_i)\prod_{m_{r',s'}\in M(r)} m_{r',s'}(h_{s'}),\end{aligned} \tag{22}$$

where

$$\begin{aligned}\bar{m}_{r\backslash s}(h_{r\backslash s}) &= \prod_{\{r',s'\}\in M(r)\backslash M(s)} m_{r',s'}(h_{s'}),\\ \bar{m}_{r,s}(h_s) &= \prod_{\{r',s'\}\in M(r,s)} m_{r',s'}(h_{s'}),\\ \overline{\Psi}(h_r, x_{r\backslash s}) &= \prod_{i,j\in r}\Psi(h_i,h_j)\prod_{i\in r\backslash s}\Phi(h_i,x_i).\end{aligned}$$

The $M(r)$ denotes the indices of messages $m_{r',s'}$ that $s'\in sub(r)\cup\{r\}$, and $r'\setminus s'$ is outside $r$. $M(r,s)$ is the set of indices of messages $m_{r',s'}$ where $r'\in sub(r)\setminus s$ and $s'\in sub(s)\cup\{s\}$.

With the injective embedding assumption for each message $\widetilde{\nu}_{r,s} = \int\phi(h_s)m_{r,s}(h_s)dh_s$ and $\widetilde{\mu}_r = \int\phi(h_r)q_r(h_r)dh_r$, following the reasoning (3) and (4), we can express the embeddings as

$$\widetilde{\nu}_{r,s} = \widetilde{\mathcal{T}}_1 \circ \left(x_{r\backslash s}, \{\widetilde{\nu}_{r',s'}\}_{M(r)\backslash M(s), M(r,s)}\right), \tag{23}$$

$$\widetilde{\mu}_{r} = \widetilde{\mathcal{T}}_2 \circ \left(x_{r}, \{\widetilde{\nu}_{r',s'}\}_{M(r)}\right). \tag{24}$$

Following the same parameterization in loopy BP, we represent the embeddings by neural network with rectified linear units,

$$\widetilde{\nu}_{r,s} = \sigma\Big(\sum_{i\in r} W_1^i x_r^i + W_2 \sum_{M(r)\backslash M(s)} \widetilde{\nu}_{r',s'} + W_3 \sum_{M(r,s)} \widetilde{\nu}_{r',s'}\Big) \tag{25}$$

$$\widetilde{\mu}_i = \sigma\Big(\sum_{i\in r} W_4^i x_i + W_5 \sum_{M(r)} \widetilde{\nu}_{r',s'}\Big) \tag{26}$$

where $\mathbf{W} = \{\{W_1^i\}, W_2, W_3, \{W_4^i\}, W_5\}$ are matrices with appropriate sizes. The generalized BP embedding updates will be almost the same as Algorithm 2 except the order of the iterations. We start from the messages into the smallest region first (Yedidia et al., 2001b).

**Remark:** The choice of basis clusters and the form of messages determine the dependency in the embedding. Please refer to Yedidia et al. (2005) for details about the principles to partition the graph structure, and several other generalized BP variants with different messages forms. The algorithms proposed for minimizing the Bethe free energy (Minka, 2001; Heskes, 2002; Yuille, 2002) can also be extended for Kikuchi free energy, resulting in different embedding forms.

# E Derivatives Computation in Algorithm 3

We can use the chain rule to obtain the derivatives with respect to $\mathbf{U}^T = \{\mathbf{W}^T, \mathbf{u}^T\}$. According to Equation 16 and Equation 17, the message passed to supervised label $y_n$ for $n-$th sample can be represented as $m_y^n = \sum_{i\in\mathcal{V}} \tilde{\mu_i}^n$, and the corresponding derivative can be denoted as

$$\frac{\partial l}{\partial m_y^n} = \frac{\partial l}{\partial f}\frac{\partial f}{\partial \sigma(m_y^n)}\frac{\partial \sigma(m_y^n)}{\partial m_y^n}$$

The term $\frac{\partial l}{\partial f}$ depends on the supervised information and the loss function we used, and $\frac{\partial l}{\partial \sigma(m_y^n)} = \mathbf{u}^T\frac{\partial l}{\partial f}$. The last term $\frac{\partial \sigma(m_y^n)}{\partial m_y^n}$ depends on the nonlinear function $\sigma$ we used here.

The derivatives with respect to $\mathbf{u}$ for the current encountered sample $\{\chi_n, y_n\}$ SGD iteration are

$$\nabla_{\mathbf{u}} l(f(\widetilde{\mu}^n; \mathbf{U}), y_n) = \frac{\partial l}{\partial f}\sigma(m_y^n)^T \tag{27}$$

In order to update the embedding parameters $\mathbf{W}$, we need to obtain the derivatives with respect to the embedding of each hidden node, i.e., $\frac{\partial l}{\partial \tilde{u_i}^n} = \frac{\partial l}{\partial m_y^n}, \forall i \in \mathcal{V}$.

**Embedded Mean Field**

In mean field embedding, we unfold the fixed point equation by the iteration index $t = 1, 2, \ldots, T$. At $t-$th iteration, the partial derivative is denoted as $\frac{\partial l}{\partial \tilde{\mu_i}^{n(t)}}$. The partial derivative with respect to the embedding obtained by last round fixed point iteration is already defined above: $\frac{\partial l}{\partial \tilde{\mu_i}^{n(T)}} = \frac{\partial l}{\partial m_y^n}$

Then the derivatives can be obtained recursively: $\frac{\partial l}{\partial \tilde{\mu_i}^{n(t)}} = \sum_{j, i\in\mathcal{N}(j)} W_2^T \frac{\partial l}{\partial \tilde{\mu_j}^{n(t+1)}}\frac{\partial \sigma}{\partial (W_1 x_j + W_2 l_j^{(t)})}$, $t = 1, 2, \ldots, T-1$. Similarly, the parameters $\mathbf{W}$ are also updated cumulatively as below.

$$\frac{\partial l}{\partial (W_1 x_i + W_2 l_i^{(t)})} = \sum_{j, i\in\mathcal{N}(j)} \frac{\partial l}{\partial \tilde{\mu_j}^{n(t+1)}}\frac{\partial \sigma}{\partial (W_1 x_j + W_2 l_j^{(t)})} \tag{28}$$

$$\nabla_{W_1} l(f(\widetilde{\mu}^n; \mathbf{U}), y_n) = \sum_{i\in\mathcal{V}_n}\sum_{t=1}^{T-1} \frac{\partial l}{\partial (W_1 x_i + W_2 l_i^{(t)})} x_i^T \tag{29}$$

$$\nabla_{W_2} l(f(\widetilde{\mu}^n; \mathbf{U}), y_n) = \sum_{i\in\mathcal{V}_n}\sum_{t=1}^{T-1} \frac{\partial l}{\partial (W_1 x_i + W_2 l_i^{(t)})} l_i^{(t)T} \tag{30}$$

**Embedding Loopy BP**

Similar as above case, we can first obtain the derivatives with respect to embeddings of hidden variables $\frac{\partial l}{\partial \tilde{\mu_i}^n} = \frac{\partial l}{\partial m_y^n}$. Since the last round of message passing only involves the edge-to-node operations, we can easily get the following derivatives.

$$\nabla_{W_3} l(f(\widetilde{\mu}^n;\mathbf{U}),y_n) = \sum_{i\in\mathcal{V}} \frac{\partial l}{\partial \tilde{\mu_i}^n} \frac{\partial\sigma}{\partial(W_3x_i+W_4\sum_{k\in\mathcal{N}(i)}\widetilde{\nu}_{ki}^{n(T)})} x_i^T \tag{31}$$

$$\nabla_{W_4} l(f(\widetilde{\mu}^n;\mathbf{U}),y_n) = \sum_{i\in\mathcal{V}} \frac{\partial l}{\partial \tilde{\mu_i}^n} \frac{\partial\sigma}{\partial(W_3x_i+W_4\sum_{k\in\mathcal{N}(i)}\widetilde{\nu}_{ki}^{n(T)})} (\sum_{k\in\mathcal{N}(i)}\widetilde{\nu}_{ki}^{n(T)})^T \tag{32}$$

$$\tag{33}$$

Now we consider the partial derivatives for the pairwise message embeddings for different $t$. Again, the top level one is trivial, which is given by $\frac{\partial l}{\partial \widetilde{\nu_{ij}}^{n(T)}} = W_4^T \frac{\partial l}{\partial \tilde{\mu_j}} \frac{\partial\sigma}{\partial(W_3x_j+W_4\sum_{k\in\mathcal{N}(j)}\widetilde{\nu}_{kj}^{(T)})}$. Using similar recursion trick, we can get the following chain rule for getting partial derivatives with respect to each pairwise message in each stage of fixed point iteration.

$$\frac{\partial l}{\partial\widetilde{\nu}_{ij}^{n(t)}} = \sum_{p\in\mathcal{N}(j)\setminus i} W_2^T \frac{\partial l}{\partial\widetilde{\nu}_{jp}^{n(t+1)}} \frac{\partial\sigma}{\partial(W_1x_j+W_2\sum_{k\in\mathcal{N}(j)\setminus p}[\widetilde{\nu}_{kj}^{n(t)}])} \tag{34}$$

Then, we can update the parameters $W_1, W_2$ using following gradients.

$$\nabla_{W_1} l(f(\widetilde{\mu}^n;\mathbf{U}),y_n) = \sum_{t=1}^{T-1}\sum_{(i,j)\in\mathcal{E}_n} \frac{\partial l}{\partial\widetilde{\nu}_{ij}^{n(t+1)}} \frac{\partial\sigma}{\partial(W_1x_i+W_2\sum_{k\in\mathcal{N}(i)\setminus j}[\widetilde{\nu}_{ki}^{n(t)}])} x_i^T \tag{35}$$

$$\nabla_{W_2} l(f(\widetilde{\mu}^n;\mathbf{U}),y_n) = \sum_{t=1}^{T-1}\sum_{(i,j)\in\mathcal{E}_n} \frac{\partial l}{\partial\widetilde{\nu}_{ij}^{n(t+1)}} \frac{\partial\sigma}{\partial(W_1x_i+W_2\sum_{k\in\mathcal{N}(i)\setminus j}[\widetilde{\nu}_{ki}^{n(t)}])} (\sum_{k\in\mathcal{N}(i)\setminus j}[\widetilde{\nu}_{ki}^{n(t)}])^T \tag{36}$$

$$\tag{37}$$