Journal of Machine Learning Research 24 (2023) 1-54 Submitted 4/21; Revised 3/23; Published 6/23

Factor Graph Neural Networks

Zhen Zhang* ZHEN.ZHANG02@ADELAIDE.EDU.AU
Australian Institute for Machine Learning &
University of Adelaide, Australia

Mohammed Haroon Dupty* DMHAROON@COMP.NUS.EDU.SG
National University of Singapore, Singapore

Fan Wu FANW6@ILLINOIS.EDU
University of Illinois at Urbana-Champaign, USA

Javen Qinfeng Shi JAVEN.SHI@ADELAIDE.EDU.AU
Australian Institute for Machine Learning &
University of Adelaide, Australia

Wee Sun Lee LEEWS@COMP.NUS.EDU.SG
National University of Singapore, Singapore

Editor: Tommi Jaakkola

Abstract

In recent years, we have witnessed a surge of Graph Neural Networks (GNNs), most of which can learn powerful representations in an end-to-end fashion with great success in many real-world applications. They have resemblance to Probabilistic Graphical Models (PGMs), but break free from some limitations of PGMs. By aiming to provide expressive methods for representation learning instead of computing marginals or most likely configurations, GNNs provide flexibility in the choice of information flowing rules while maintaining good performance. Despite their success and inspirations, they lack efficient ways to represent and learn higher-order relations among variables/nodes. More expressive higher-order GNNs which operate on k-tuples of nodes need increased computational resources in order to process higher-order tensors. We propose Factor Graph Neural Networks (FGNNs) to effectively capture higher-order relations for inference and learning. To do so, we first derive an efficient approximate Sum-Product loopy belief propagation inference algorithm for discrete higher-order PGMs. We then neuralize the novel message passing scheme into a Factor Graph Neural Network (FGNN) module by allowing richer representations of the message update rules; this facilitates both efficient inference and powerful end-to-end learning. We further show that with a suitable choice of message aggregation operators, our FGNN is also able to represent Max-Product belief propagation, providing a single family of architecture that can represent both Max and Sum-Product loopy belief propagation. Our extensive experimental evaluation on synthetic as well as real datasets demonstrates the potential of the proposed model.

Keywords: Graphical Models, Belief Propagation, Graph Neural Networks

*. Equal contribution.

1. Introduction

Deep neural networks are powerful function approximators that have been extremely successful in practice. While fully connected networks are universal approximators, successful networks in practice tend to be structured, *e.g.*, grid-structured convolutional neural networks and chain-structured gated recurrent neural networks (*e.g.*, LSTM, GRU). Graph neural networks (Gilmer et al., 2017; Xu et al., 2018; Yoon et al., 2019) have recently been successfully used with graph-structured data to capture pairwise dependencies between variables and to propagate the information to the entire graph.

GNNs learn node representations by iteratively passing messages between nodes within their neighbourhood and updating the node embeddings based on the messages received. Though successful, these models are limited by the first order approximations they make in aggregating information from the neighbouring nodes. The dependencies in the real-world data are often of higher-order which cannot be captured by only pairwise modeling. For example, in LDPC encoding, the bits of a signal are grouped into several clusters and in each cluster, the parity of all bits is constrained to be equal to zero (Zarkeshvari and Banihashemi, 2002). For good performance, these higher-order constraints should be exploited in the decoding procedure. Furthermore, many naturally occurring graphs like molecules exhibit repeating substructures like motifs; atoms in a molecule additionally satisfy higher-order valence constraints (Wu et al., 2018; Agarwal et al., 2006). We can learn better node representations if we can design better message passing schemes that can directly utilize the higher-order dependencies that are not captured using pair-wise dependencies.

Node interactions have also been modeled by Probabilistic Graphical Models (PGMs) (Wainwright and Jordan, 2008; Koller and Friedman, 2009) wherein nodes are viewed as random variables with a factorized joint distribution defined over them. Research that focuses on such models has been directed towards finding approximate inference algorithms to compute node marginals or the most likely configuration of the variables. There is rich literature of theoretically grounded PGM inference algorithms to find the state of a node given its statistical relations with other variable nodes in the graph. Knowledge of such inference algorithms can provide good inductive bias if it can be encoded in the neural network. The inductive bias lets the network favour certain solutions over others, and if the bias is indeed consistent with the target task, it helps in better generalization (Battaglia et al., 2018). So, while we use deep learning methods to learn representations in an end-to-end setting, better generalization may be achievable for some tasks with the inductive bias of classical inference algorithms. Unfortunately, these approximate inference algorithms become inefficient at higher-order.

One such well-known algorithm to find approximate node marginals is Loopy Belief Propagation (LBP) (Murphy et al., 2013; Pearl, 2014) which operates on the factor graph data structure. A factor graph is a bipartite graph with a set of variable nodes connected to a set of factor nodes; each factor node indicates the presence of dependencies among its connected variables. In this paper, we propose to leverage Sum-Product LBP to formulate the message passing updates and build a better graph representation learning model. To this end, we derive an efficient message passing algorithm based on LBP with arbitrary higher-order factors on discrete graphical models, with the assumption that higher-order factors are of low rank, parameterized in the form of a mixture of rank-1 tensors. The derived

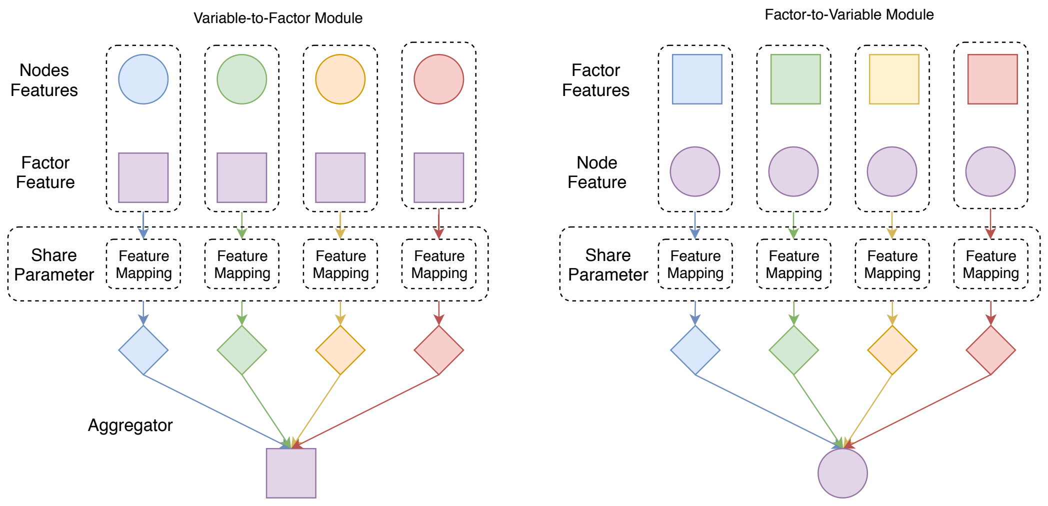


Figure 1: The structure of the Factor Graph Neural Network (FGNN): the Variable-to-Factor (VF) module is shown on the left and the Factor-to-Variable (FV) module is shown on the right.

message passing updates only need two operations, matrix multiplication and Hadamard product, and their complexity grows linearly with the number of variables in the factor. Furthermore, this parameterization can represent any factor exactly with a large enough set of rank-1 tensors; the number of rank-1 tensors required can grow exponentially for some problems but often in practice, a small number is sufficient for a good approximation.

Further, we represent the message passing updates in a neural network module and unroll the inference over discrete graphical models as a computational graph. We allow the messages to be arbitrary real valued vectors (instead of being constrained to be positive as in LBP) and treat the messages as latent vectors in a network; the latent vectors produced by the network can then be used for learning the target task through end-to-end training. Instead of using just the product operations to aggregate the set of latent vectors, we allow the use of other aggregrators, potentially even universal set functions, to provide more flexibility in representation learning. We refer to the process of unrolling the algorithm, relaxing some of the constraints, modifying some of the components to potentially make the network more powerful, and using the resulting network as a component for end-to-end learning as *neuralizing* the algorithm. We call the neural module as a Factor Graph Neural Network (FGNN).

The FGNN is defined using two types of modules, the Variable-to-Factor (VF) module and the Factor-to-Variable (FV) module (see Figure 1). These modules are combined into a layer, and the layers are stacked together into an algorithm. Though the FGNN is motivated with Sum-Product LBP, we show that by using a different form of low-rank tensor representation and aggregator function, it is able to exactly parameterize the Max-Product Belief Propagation, which is a widely used approximate *maximum a posteriori* (MAP) inference algorithm for PGMs, as well. Theoretically, this shows that FGNN can represent both Max-Product and Sum-Product within a single architecture simply by changing the aggregator function; furthermore if a universal approximator is used as the aggregator, it would be able to learn to approximate the better of the two message passing algorithms on the problem.

The theoretical relationship with Sum-Product and Max-Product provides understanding on the representational capabilities of GNNs in general, and of FGNN in particular, *e.g.* it can solve problems solvable by graphical model message passing algorithms *e.g.*, (Bayati et al., 2008; Kim and Pearl, 1983). From the practical perspective, the factor graph provides a flexible way for specifying dependencies among the variables, including higher-order dependencies. Furthermore, inference algorithms for many types of graphs, *e.g.*, graphs with typed edges or nodes, are easily developed using the factor graph representation. Edges, or more generally factors, can be typed by tying together parameters of factors of the same type, or can also be conditioned from input features by making the edge or factor parameters a function of the features; nodes can similarly have types or features with the use of factors that depend on a node variable. With typed or conditioned factors, the factor graph can also be assembled dynamically for each graph instance. FGNN provides a flexible learnable architecture for exploiting these graphical structures—just as factor graph allows easy specification of different types of PGMs, FGNN allows easy specification of both typed and conditioned variables and dependencies as well as a corresponding data-dependent approximate inference algorithm.

To be practically useful, the FGNN architecture needs to be practically *learnable* without being trapped in poor local minima. We performed experiments to explore the practical potential of FGNN on both problems where the graphical model inference aspects are clear, as well as problems where the FGNN is used mostly as a graph neural network that allows the structure of the higher-order interactions to be specified and exploited. On problems closely related to PGM inference, we experimented with a synthetic higher-order PGM inference problem, LDPC decoding, as well as a graph matching problem. FGNN performed well on the synthetic PGM inference and outperforms the standard LDPC decoding method under some noise conditions. On the graph matching problem, FGNN outperforms both a graphical model approximate inference algorithm as well as a graph neural network. To show that FGNN can be used purely as a graph neural network to exploit higher-order dependencies, we conducted experiments on handwritten character recognition within words where there is strong correlation in the character sequences, human motion prediction where different joint positions are constrained by the human body structure and molecular property prediction where higher-order correlations in the molecular graphs are likely to be present. We demonstrate that FGNN is able to exploit the higher-order information with state-of-the-art results on human motion prediction. Furthermore, FGNN is able to outperform other recent k-order GNNs (Morris et al., 2019; Maron et al., 2019) substantially on two challenging large molecular datasets (QM9 and Alchemy).

2. Related work

Belief Propagation (BP) inference algorithms have been used in variety of applications spanning computer vision, natural language processing and other machine learning domains. Due to the intractability of the algorithm in its higher-order form, first-order pairwise BP has been predominantly used in most problems. However, multiple works have proposed approaches to efficiently run higher-order BP in various settings (Lan et al., 2006; Potetz and Lee, 2008; Kohli and Kumar, 2010). Most of the approaches are designed either with assumptions on input graph structures or with restrictions on the kind of higher-order functions being modelled. Lan et al. (2006) used an adaptive state space to handle the increased complexity for higher-order 2x2 MRF clique structures. Potetz and Lee (2008) used specific assumption of linear constraints on higher-order functions to efficiently run the BP equations. Kohli and Kumar (2010) used linear envelope approximations to model the higher-order functions and showed its usefulness for semantic segmentation.

The inspiration for our low-rank formulation of higher-order functions comes from Wrigley et al. (2017) where higher-order potentials were decomposed with CP decompositions to efficiently run the sampling-based junction-tree algorithm. They used CP tensor decomposition to efficiently sample from higher-order functions in the junction-tree updates, whereas in our work, we assume tensor decomposition form of higher-order function and derive efficient message update equations for the LBP algorithm. Furthermore, our method is a deterministic approximation of LBP and we neuralize it to learn the parameters in an end-to-end training.

There have been a significant number of works which have tried to incorporate inference algorithms in deep networks (Zheng et al., 2015; Chen et al., 2015; Lin et al., 2016, 2015; Tamar et al., 2016; Karkus et al., 2017; Xu et al., 2017). A large number of these works focus on learning pairwise potentials of graphical models on top of CNN to capture relational structures among their outputs for structured prediction. These methods are largely focused on modeling for a specific task like semantic segmentation (Zheng et al., 2015; Lin et al., 2016), scene graph prediction (Xu et al., 2017), and image denoising (Wu et al., 2016). On the other hand, (Tamar et al., 2016; Karkus et al., 2017) represent planning algorithms as neural network layers in sequential decision making tasks.

Various graph neural network models have been proposed for graph structured data, including methods based on the graph Laplacian (Bruna et al., 2013; Defferrard et al., 2016; Kipf and Welling, 2016), gated networks (Li et al., 2015), and various other neural networks structures for updating the information (Duvenaud et al., 2015; Battaglia et al., 2016; Kearnes et al., 2016; Schütt et al., 2017). Gilmer et al. (2017) show that these methods can be viewed as applying message passing on pairwise graphs and are special cases of Message Passing Neural Networks (MPNNs).

There has been some recent work on extending the graph convolutional neural networks to hyper-graphs in order to capture higher-order information (Feng et al., 2019; Yadati et al., 2019; Jiang et al., 2019; Zhang et al., 2019). Feng et al. (2019); Jiang et al. (2019) used clique-expansion of hyper-edges to extend convolutional operation to hyper-graphs. Such modeling is equivalent to decomposing an hyper-edge to a set of pairwise edges. Similar approximation is applied in Yadati et al. (2019) where the number of pairwise edges added are reduced and are linearly dependent on the size of the hyperedge. Although, these methods operate on hyper-graphs, effectively the hyper-graphs are reduced to graphs with pairwise edges.

Recently, Morris et al. (2019) and Maron et al. (2019) used Weisfeiler Lehman (WL) graph isomorphism tests to construct increasingly powerful GNNs. They proposed models of message passing called k-order GNNs to capture higher-order structures and compared their expressiveness with higher-order WL tests. In contrast to k-order GNNs which build on graph isomorphism testing, FGNN builds on probabilistic graphical models, which provide a rich modeling language allowing the designer to specify prior knowledge in the form of pairwise as well as higher-order dependencies in a factor graph. As k-order GNNs are theoretically shown to be more expressive in capturing higher-order information, we compare our model with the k-order GNNs on the molecular datasets.

Running inference algorithms on graph neural networks has been previously explored in Yoon et al. (2019); Dai et al. (2016) and Chen et al. (2018). These works showed methods of running graphical model inference with GNN message passing updates. However, they operate with the assumption of pairwise graphical models and not more general higher-order models. In contrast, our work deals mainly with running the inference algorithms on the higher-order graphical model as a graph neural network. Following our initial work, Satorras and Welling (2020) proposed a graph neural network based on the factor graph for LDPC code decoding that also exploits the loopy belief propagation messages. However they did not connect message passing on factor graphs with higher-order factors represented using tensor decompositions as we have done which further gives a theoretical understanding of the relation between inference algorithms and GNNs.

3. Preliminaries

In this section, we briefly review two concepts central to our approach, low rank tensor decomposition and the loopy belief propagation inference algorithm.

3.1 Tensor decompositions

Tensors are generalizations of matrices to higher dimensions. An order-m tensor $\mathbf{T}$ is an element in $\mathbb{R}^{N_1 \times N_2 \cdots \times N_m}$ with N_k possible values in k^{th} dimension for $k \in \{1, 2, \ldots, m\}$. Tensor rank decompositions provide succinct representation of tensors. In CANDECOMP / PARAFAC (CP) decomposition, a tensor $\mathbf{T}$ can be represented as a linear combination of outer products of vectors as

$$\mathbf{T} = \sum_{r=1}^{R} \lambda^r w_1^r \otimes w_2^r \otimes \cdots \otimes w_m^r \tag{1}$$

where $\lambda^r \in \mathbb{R}$, $w_k^r \in \mathbb{R}^{N_k}$, $\otimes$ is the outer product operator, *i.e.*, $\mathbf{T}(i_1, i_2 \ldots, i_m) = \sum_{r=1}^{R} \lambda^r w_{1_{i_1}}^r w_{2_{i_2}}^r \cdots w_{m_{i_m}}^r$, and the term $w_1^r \otimes w_2^r \otimes \cdots \otimes w_m^r$ is a rank-1 tensor. The scalar coefficients λ^r can optionally be absorbed into $\{w_k^r\}$. The smallest R for which an exact R-term decomposition exists is the rank of tensor $\mathbf{T}$ and the decomposition (1) is its R-rank approximation. With this compact representation, an exponentially large tensor $\mathbf{T}$ with $N_1 \times N_2 \cdots \times N_m$ entries can be represented with R vectors for each variable in $\mathbf{T}$, *i.e.*, with a total of $R(N_1 + N_2 + \cdots + N_m)$ parameters. More information about tensor decompositions can be found in Kolda and Bader (2009), and Rabanser et al. (2017).

3.2 Graphical models and Loopy belief propagation

Probabilistic Graphical Models (PGMs) use graphs to model dependencies among random variables. These dependencies are conveniently represented using a factor graph, which is a bipartite graph $\mathcal{G} = (\mathcal{V}, \mathcal{C}, \mathcal{E})$ where each vertex $i \in \mathcal{V}$ in the graph is associated with a random variable $x_i \in \mathbf{x}$, each vertex $c \in \mathcal{C}$ is associated with a non-negative function ϕ_c, and an edge connects a variable vertex i to a factor vertex c if ϕ_c depends on x_i. An example factor graph is shown in Figure 2.

We consider PGMs which model dependencies between discrete random variables. Let $\mathbf{x}$ be the set of all variables and let $\mathbf{x}_c$ be the subset of variables that ϕ_c depends on. The joint distribution of variables factorizes over $\mathcal{C}$ as

$$P(\mathbf{x}) = \frac{1}{Z}\prod_{c\in\mathcal{C}}\phi_c(\mathbf{x}_c) \qquad\qquad Z = \sum_{\mathbf{x}}\prod_{c\in\mathcal{C}}\phi_c(\mathbf{x}_c) \tag{2}$$

Z is the normalizing constant. Without loss of generality, we assume all variables can take d values and consequently $\phi_c(\mathbf{x}_c) \in \mathbb{R}^{d^{|\mathbf{x}_c|}}$.

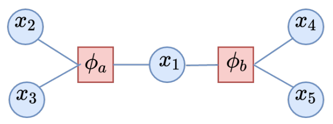


Figure 2: A factor graph where ϕ_a depends on $\{x_1, x_2, x_3\}$ while ϕ_b depends on $\{x_1, x_4, x_5\}$.

Marginal and MAP Inference In this paper we consider two main inference tasks — the *marginal* inference and the *maximum a posteriori* (MAP) inference. The aim of marginal inference is to compute the marginals $p_i(x_i)$

$$p_i(x_i) = \sum_{\mathbf{x}_{\mathcal{V}\setminus\{i\}}} P(\mathbf{x}) = \sum_{\mathbf{x}_{\mathcal{V}\setminus\{i\}}} \frac{1}{Z}\prod_{c\in\mathcal{C}}\phi_c(\mathbf{x}_c), \tag{3}$$

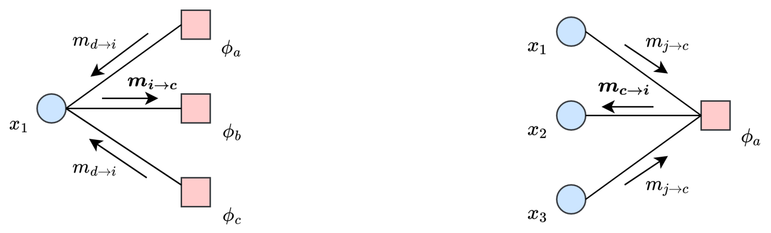
and the aim of MAP inference is to find the assignment which maximizes $P(\mathbf{x})$, that is

$$\begin{aligned}\mathbf{x}^* &= \operatorname*{argmax}_{\mathbf{x}}\prod_{c\in\mathcal{C}}\phi_c(\mathbf{x}_c)\\ &= \operatorname*{argmax}_{\mathbf{x}}\sum_{c\in\mathcal{C}}\log\phi_c(\mathbf{x}_c)\end{aligned} \tag{4}$$

Loopy Belief Propagation The marginal inference and MAP inference problem are NP-hard in general, and thus approximation algorithms are usually required. Different versions of the loopy belief propagation (LBP) algorithms (Pearl, 2014; Murphy et al., 2013) compute approximate marginals $p(x_i)$ at each node x_i, or the approximate MAP assignment, by sending messages between factor and variables nodes on a factor graph. First we introduce the Sum-Product loopy belief propagation. Essentially, the Sum-Product LBP starts by initializing two kinds of messages, factor-to-variable $m_{c\to i}(x_i)$ and variable-to-factor $m_{i\to c}(x_i)$. Messages are a function of the variable in the variable node, updated with the following recursive equations,

$$m_{i\to c}(x_i) = \prod_{d\in N(i)\setminus\{c\}} m_{d\to i}(x_i) \tag{5}$$

$$m_{c\to i}(x_i) = \sum_{\mathbf{x}_c\setminus\{x_i\}}\phi_c(\mathbf{x}_c)\prod_{j\in N(c)\setminus\{i\}} m_{j\to c}(x_j) \tag{6}$$



(a) Variable to Factor message (b) Factor to Variable message

Figure 3: Loopy Belief Propagation messages

where $N(i)$ is the set of neighbours of i and messages $m_{i\to c}, m_{c\to i} \in \mathbb{R}^d$. As illustrated in Figure 3, $m_{i\to c}$ is the message from variable i to factor c and $m_{c\to i}$ is the message from factor c to variable i. After sufficient number of iterations, the belief of variables is computed by

$$b_i(x_i) = \prod_{c\in N(i)} m_{c\to i}(x_i). \tag{7}$$

Similarly, the Max-Product belief propagation algorithm can be formulated as

$$m_{i\to c}(x_i) = \prod_{d\in N(i)\backslash\{c\}} m_{d\to i}(x_i) \tag{8}$$

$$m_{c\to i}(x_i) = \max_{\mathbf{x}_c\backslash\{x_i\}} \phi_c(\mathbf{x}_c) \prod_{j\in N(c)\backslash\{i\}} m_{j\to c}(x_j) \tag{9}$$

$$b_i(x_i) = \prod_{c\in N(i)} m_{c\to i}(x_i). \tag{10}$$

Performed in log space, the product operator in Eqns 8, 9, 10 becomes sum and we have the Max-Sum algorithm which may be better behaved numerically.

4. Proposed Method

In this section, we derive the FGNN model through neuralizing the Sum-Product loopy belief propagation that utilizes the low rank decomposition of higher-order potentials. Then we show that with a slight modification the model can also mimic the Max-Sum or equivalently the Max-Product belief propagation.

4.1 Low Rank Sum-Product Loopy Belief Propagation

We start the derivation by writing the LBP equations in vectorized form. Consider a factor $\phi_c(\mathbf{x}_c)$ over n_c number of variables *i.e.* $\mathbf{x}_c = [x_1, x_2 \ldots, x_{n_c}]$. Then the message update equations are,

$$m_{i\to c} = \prod_{d\in N(i)\backslash\{c\}} m_{d\to i} \tag{11}$$

$$m_{c\to i} = \sum_{\mathbf{x}_c\backslash\{x_i\}} \phi_c(\mathbf{x}_c) \prod_{j\in N(c)\backslash\{i\}} m_{j\to c} \tag{12}$$

where $\phi_c(\mathbf{x}_c) \in \mathbb{R}^{d^{n_c}}$ and the messages $m_{j\to c}, m_{c\to i}$ are in $\mathbb{R}^d$. A tensorized way to implement Equation (12) would be to take the outer product of all incoming messages $m_{j\to c}$, expand a dimension corresponding to the dimension of the i^{th} variable and elementwise multiply with the $\phi_c(\mathbf{x}_c)$ tensor. Then, we can marginalize all variables except x_i to get the message $m_{c\to i}$ *i.e.*,

$$m_{c\to i} = \sum_{\mathbf{x}_c \setminus \{x_i\}} \phi_c(\mathbf{x}_c) \odot (m_{1\to c} \otimes \cdots \otimes \mathbf{1}_{i\to c} \otimes \cdots \otimes m_{n_c \to c}) \tag{13}$$

where $\mathbf{1}_{i\to c}$ is a vector of ones in $\mathbb{R}^d$, $\otimes$ is the outer product and $\odot$ is elementwise multiplication or Hadamard product. Note that the summation operator $\sum_{\mathbf{x}_c \setminus \{x_i\}}$ performs "marginalization" operation or summation across all dimensions except the i^{th} dimension corresponding to variable x_i which reduces the order-n_c tensor to an R^d vector.

Since $\phi_c(\mathbf{x}_c)$ is a tensor in $\mathbb{R}^{d^{n_c}}$, it can be represented as a sum of R fully factored terms in CP decomposition form (1).

$$\phi_c(\mathbf{x}_c) = \sum_{r=1}^{R} \lambda_r w^r_{c,1} \otimes w^r_{c,2} \otimes \cdots \otimes w^r_{c,n_c} \tag{14}$$

where $w^r_{c,j} \in \mathbb{R}^d$ and λ_r are real-valued scalars. This representation is efficient if ϕ_c is of low-rank i.e., $R \ll d^{n_c}$, and in that case, a d^{n_c} dimensional tensor is compressed into a set of $n_c \cdot R$ vectors of d dimensons each. Please refer Section 3.1 on Preliminaries for details on CP Tensor Decomposition.

We posit that such a low-rank representation is often a good approximation of ϕ_c in practice; the method is likely to be useful when this assumption holds. Previously, such low rank approximations have been shown to be useful in variety of real world tasks including semantic segmentation and knowledge graph embedding (Wrigley et al., 2017; Kohli and Kumar, 2010; Trouillon et al., 2017). Moreover, we provide supporting evidence on the low-rank assumption in our ablation experiments in Section 5.4 and 5.5 (See Figure 7 and 11).

Absorbing λ_r in (14) into weights $\{w^r_{c,j}\}$ and substituting in (13), we have

$$m_{c\to i} = \sum_{\mathbf{x}_c \setminus \{x_i\}} \Big(\sum_{r=1}^{R} w^r_{c,1} \otimes w^r_{c,2} \otimes \cdots \otimes w^r_{c,n_c} \Big) \odot \Big(m_{1\to c} \otimes \cdots \otimes \mathbf{1}_{i\to c} \otimes \cdots \otimes m_{n_c\to c} \Big) \tag{15a}$$

$$= \sum_{\mathbf{x}_c \setminus \{x_i\}} \sum_{r=1}^{R} (w^r_{c,1} \odot m_{1\to c}) \otimes \cdots w^r_{c,i} \cdots \otimes (w^r_{c,n_c} \odot m_{n_c \to c}) \tag{15b}$$

In Equation (15b), we have used the distribution rule *i.e.* $\forall\, u, v \in \mathbb{R}^d$ and $u', v' \in \mathbb{R}^{d'}$, then $(u \otimes u') \odot (v \otimes v') = (u \odot v) \otimes (u' \odot v')$.

The variables are grouped together with the factor parameters corresponding to them. Now we can marginalize out a variable easily as we have a sum of fully factorized functions. We simply push the outer summation inside, distribute and separately evaluate it over each of the univariate products $\big(w^r_{c,j} \odot m_{j\to c}\big)$. This gives us,

$$\begin{aligned}
m_{c\to i} =& \sum_{r=1}^{R} \sum_{\mathbf{x}_c\setminus\{x_i\}} \big(w^r_{c,1} \odot m_{1\to c}\big) \otimes \cdots w^r_{c,i} \cdots \otimes \big(w^r_{c,n_c} \odot m_{n_c\to c}\big) && \text{(16a)}\\
=& \sum_{r=1}^{R} w^r_{c,i} \Big(\sum_{x_1} w^r_{c,1} \odot m_{1\to c}\Big) \cdots \Big(\sum_{x_{n_c}} w^r_{c,n_c} \odot m_{n_c\to c}\Big) && \text{(16b)}\\
=& \sum_{r=1}^{R} w^r_{c,i}\gamma^r_{c,1}\cdots\gamma^r_{c,n_c} \qquad ;\text{with } \gamma^r_{c,i}=1 && \text{(16c)}\\
=& \sum_{r=1}^{R} w^r_{c,i}\boldsymbol{\gamma}^r_c && \text{(16d)}
\end{aligned}$$

In Equation (16a), we have swapped the summations and in Equation (16b), have distributed the summation $\sum_{\mathbf{x}_c\setminus\{x_i\}}$ over each variable. In Equation (16c), we evaluate the summation over each variable to get $\gamma^r_{c,j}$ *i.e.* $\big(\sum_{x_j} w^r_{c,j} \odot m_{j\to c}\big) = w^{r^T}_{c,j} m_{j\to c} = \gamma^r_{c,j} \in \mathbb{R}$, which is a scalar. To rule out the message from variable x_i, we set $\gamma^r_{c,i} = 1$ and therefore, the product $\boldsymbol{\gamma}^r_c = \gamma^r_{c,1}\cdot\gamma^r_{c,2}\cdots\gamma^r_{c,n_c} \in \mathbb{R}$ is also a scalar in Equation (16d).

Since $m_{c\to i}$ is a linear combination of R number of $w^r_{c,i}$ vectors in Equation (16d), we can rewrite it in matrix form. For this, we stack the R component weight vectors for each variable as matrix $\mathbf{W}_{c,i} = [w^1_{c,i}, w^2_{c,i}, \ldots, w^R_{c,i}] \in \mathbb{R}^{d\times R}$. Similarly, we stack R number of $\boldsymbol{\gamma}^r_c$'s together as vector $\boldsymbol{\Gamma}_c = [\boldsymbol{\gamma}^1_c, \boldsymbol{\gamma}^2_c, \ldots, \boldsymbol{\gamma}^R_c]^T \in \mathbb{R}^{R\times 1}$. Then, we can rewrite the Equation (16d) in matrix form as

$$m_{c\to i} = \mathbf{W}_{c,i}\boldsymbol{\Gamma}_c \tag{17}$$

Note that since $\boldsymbol{\Gamma}_c = [\boldsymbol{\gamma}^1_c, \boldsymbol{\gamma}^2_c, \ldots, \boldsymbol{\gamma}^R_c]^T \in \mathbb{R}^{R\times 1}$ where each $\boldsymbol{\gamma}^r_c$ is a product of $\gamma^r_{c,j}$'s *i.e.* $\boldsymbol{\gamma}^r_c = \gamma^r_{c,1}\cdot\gamma^r_{c,2}\cdots\gamma^r_{c,n_c}$ (from Equations (16c), (16d), we can rewrite $\boldsymbol{\Gamma}_c$ as elementwise product of vectors in each variable as:

$$\boldsymbol{\Gamma}_c = \begin{bmatrix}\boldsymbol{\gamma}^1_c\\ \boldsymbol{\gamma}^2_c\\ \vdots\\ \boldsymbol{\gamma}^R_c\end{bmatrix} = \begin{bmatrix}\gamma^1_{c,1}\cdot\gamma^1_{c,2}\cdots\gamma^1_{c,n_c}\\ \gamma^2_{c,1}\cdot\gamma^2_{c,2}\cdots\gamma^2_{c,n_c}\\ \vdots\\ \gamma^R_{c,1}\cdot\gamma^R_{c,2}\cdots\gamma^R_{c,n_c}\end{bmatrix} = \begin{bmatrix}\gamma^1_{c,1}\\ \gamma^2_{c,1}\\ \vdots\\ \gamma^R_{c,1}\end{bmatrix} \odot \begin{bmatrix}\gamma^1_{c,2}\\ \gamma^2_{c,2}\\ \vdots\\ \gamma^R_{c,2}\end{bmatrix} \cdots \odot \begin{bmatrix}\gamma^1_{c,n_c}\\ \gamma^2_{c,n_c}\\ \vdots\\ \gamma^R_{c,n_c}\end{bmatrix} \tag{18}$$

This gives us,

$$\boldsymbol{\Gamma}_c = [\Gamma_{c,1} \odot \Gamma_{c,2} \cdots \odot \Gamma_{c,n_c}] \tag{19}$$

where each $\Gamma_{c,j} = [\gamma^1_{c,j}, \gamma^2_{c,j}, \ldots, \gamma^R_{c,j}]^T \in \mathbb{R}^{R\times 1}$. Note that from Equation (16c), $\boldsymbol{\Gamma_c}$ contains $\Gamma_{c,i}$ as a vector of all ones *i.e.* $\Gamma_{c,i} = [\gamma^1_{c,i}, \gamma^2_{c,i}, \ldots, \gamma^R_{c,i}]^T = [1, 1, \ldots, 1]^T$. Therefore, Equation (17) can now be written as

$$m_{c\to i} = \mathbf{W}_{c,i}\big[\Gamma_{c,1} \odot \Gamma_{c,2} \cdots \odot \Gamma_{c,n_c}\big] \tag{20}$$

Furthermore, from Equation (16c) we have $\gamma_{c,j} = \sum_{x_j} w^r_{c,j} m_{j\to c} = {w^r_{c,j}}^T m_{j\to c}$. Therefore, we can write $\Gamma_{c,j}$ as

$$\begin{aligned}\Gamma_{c,j} &= [\gamma^1_{c,j}, \gamma^2_{c,j}, \dots, \gamma^R_{c,j}]^T && (21a)\\ &= [w^1_{c,j}, w^2_{c,j}, \dots, w^R_{c,j}]^T m_{j\to c} && (21b)\\ &= \mathbf{W}^T_{c,j} m_{j\to c} && (21c)\end{aligned}$$

Now, combining with Equation (20) we have new message passing updates for the low-rank loopy belief propagation algorithm,

$$m_{c\to i} = \mathbf{W}_{c,i}\Big[(\mathbf{W}^T_{c,1} m_{1\to c}) \odot (\mathbf{W}^T_{c,2} m_{2\to c}) \cdots \odot (\mathbf{W}^T_{c,n_c} m_{n_c\to c})\Big]_{\setminus i} \quad (22a)$$

$$m_{i\to c} = \bigodot_{d\in N(i)\setminus\{c\}} m_{d\to i} \quad (22b)$$

Belief update is simple. For the variable x_i, we project the messages from the other variables sharing a factor with x_i to $\mathbb{R}^R$, perform elementwise multiplication and then project the product back to $\mathbb{R}^d$. Thereafter, multiply such messages from all the factors x_i is connected to get the updated belief of x_i.

Clearly, the computational complexity of the message updates grows linearly with the addition of variables to factors and thereby the algorithm is efficent enough to run with higher-order factors.

4.2 Neuralizing Low Rank Sum-Product LBP

To learn better node and consequently, graph representations in an end-to-end setting, we seek to neuralize the Low Rank LBP algorithm by writing the message passing updates as a functionally equivalent neural network module and *unroll* the inference algorithm as a computational graph. We further replace the positive message vectors in LBP with unconstrained real valued hidden latent vector representations initialized from a feature extractor network.

In the Low-rank LBP algorithm, a factor is parameterized by the set of matrices $\{\mathbf{W}_{c,j}\}$ i.e. it has as many $\mathbf{W}_{c,j}$'s as the number of variables adjacent to it in the factor graph. We can relax this constraint and maintain $2|\{\mathbf{W}_{c,j}\}|$ matrices at each factor, one set is used for transformation of messages before Hadamard product and the other set after the product in equation (22a). The additional parameters are helpful as the two message updates are not tied with shared parameters and can be run parallelly. Also, with more parameters, this may increase the representative power of the neural-network while still being able to represent equation (22a) as the extra set of matrices can be learnt to be same as the first set.With this relaxation, we can push the outer $\mathbf{W}_{c,i}$ to equation (22b) and rewrite the LBP updates of (22a) and (22b) as,

$$m_{c\to i} = \bigodot_{j\in N(c)\setminus\{i\}} \mathbf{W}^T_{c,j} m_{j\to c} \quad (23a)$$

$$m_{i\to c} = \bigodot_{d\in N(i)\setminus\{c\}} \mathbf{W}_{d,i} m_{d\to i} \quad (23b)$$

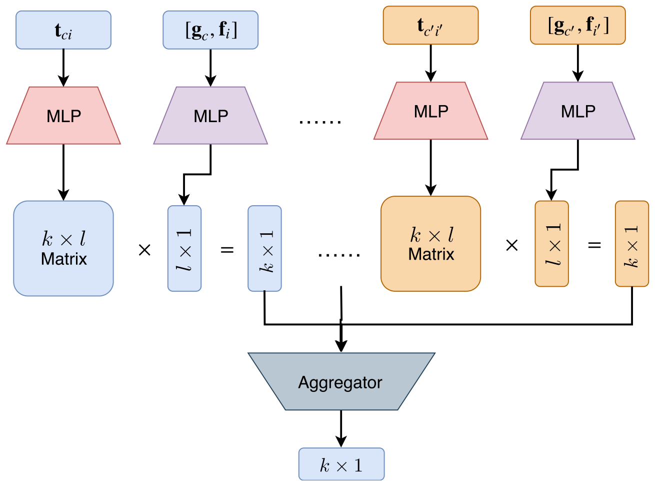


Figure 4: A single aggregator of the FGNN architecture.

The message updates of (23a) and (23b) only involve operations like matrix-vector product and elementwise multiplication. These operations can very well be represented in a neural network for end-to-end learning. But the multiplication of several terms can lead to numerical instability in learning due to overflow and underflow errors. Therefore, we replace the Hadamard product with other generalized set-function aggregators. It has been shown that given a set of input vectors, an MLP followed by *sum* (Zaheer et al., 2017) or *max* (Qi et al., 2017) aggregator is a universal set approximator; it can approximate any non-linear set function and hence can be used in place of Hadamard product, approximating the Hadamard product if necessary, and providing the capability to possibly learn a better aggregator than the Hadamard product. To provide even more approximation capabilities, we allow a multi-layer perceptron (MLP) to transform the message after aggregation just like typical graph neural networks.

A single aggregator of FGNN is shown in Figure 4. For a message between node c and node i, we optionally allow both the the latent vectors from c and i to be used as input to the MLP at node c. The matrix $\mathbf{W}$ that is used to multiply the message can be learned directly, if necessary. Alternatively, if some feature t_{ci} is available, the $\mathbf{W}_{c,i}$ can be conditioned on t_{ci} by using a MLP to output the matrix conditioned on the feature.

For implementing Equations (23a) and (23b), the nodes in Figure 4 correspond to messages $m_{c\to i}$ and $m_{i\to c}$. If we want to use the architecture as a graph neural network instead, it is convenient to define the latent vectors to correspond to factor and variable nodes as it would result in a smaller network. This can be done if we do not exclude the vector from the target node in the aggregration operation, i.e. use $m_c = \bigodot_{j\in N(c)} \mathbf{W}_{c,j}^T m_j$ and $m_i = \bigodot_{d\in N(i)} \mathbf{W}_{d,i} m_d$ as the starting points for neuralization instead. The original belief propagation equations are exact inference algorithms when there are no loops. For correctness, the information in each message is only used once for computing a node marginal. As a graph neural network, the network is trained end-to-end and can be trained to account for the repeated information. We find experimentally that the results are similar when we use networks where nodes correspond to the belief propagation messages and networks where the nodes correspond to variables and factors. Hence, we use the networks where nodes correspond to variables and factors except for theoretical results where we are simulating the Sum-Product and Max-Product algorithms.

4.3 Factor Graph Neural Network

We now describe the Factor Graph Neural Network (FGNN), a graph neural network model based on the derived low-rank LBP equations. It consists of two modules, Variable-to-Factor module and Factor-to-Variable module. Given a factor graph $\mathcal{G} = (\mathcal{V}, \mathcal{C}, \mathcal{E})$, unary features $[\mathbf{f}_i]_{i\in\mathcal{V}}$ and factor features $[\mathbf{g}_c]_{c\in\mathcal{C}}$, assume that for each edge $(c, i) \in \mathcal{E}$, with $c \in \mathcal{C}, i \in \mathcal{V}$, there is an associated edge feature vector $[t_{ci}]$. Let k and l be hidden dimensions of variable and factor node embeddings, respectively. Then, the pseudo code for a FGNN layer on $\mathcal{G}$ is shown in Algorithm 1, where $[\Phi_{\mathrm{VF}}, \Theta_{\mathrm{VF}}]$ are parameters for the Variable-to-Factor module, and $[\Phi_{\mathrm{FV}}, \Theta_{\mathrm{FV}}]$ are parameters for the Factor-to-Variable module. The factor to variable messages are computed as $\tilde{\mathbf{f}}_i = \underset{c:(c,i)\in\mathcal{E}}{\mathbf{AGG}} \mathcal{Q}(\mathbf{t}_{ci} \,|\Phi_{\mathrm{FV}})\, \mathcal{M}([\ \mathbf{g}_c, \mathbf{f}_i]|\Theta_{\mathrm{FV}})$ where $\mathcal{M}$ is a neural network mapping feature vectors to a length-k feature vector, and $\mathcal{Q}(\mathbf{e}_{ij})$ is a neural network mapping $\mathbf{e}_{ij}$ to a $k \times l$ matrix. Then by matrix multiplication and aggregation operator $\mathbf{AGG}$, a new length-l variable feature $\tilde{\mathbf{f}}_i$ is generated. Similarly, the variable to factor messages are computed as $\tilde{\mathbf{g}}_c = \underset{i:(c,i)\in\mathcal{E}}{\mathbf{AGG}} \mathcal{Q}(\mathbf{t}_{ci} \,|\Phi_{\mathrm{VF}})\, \mathcal{M}([\mathbf{g}_c, \mathbf{f}_i]|\Theta_{\mathrm{VF}})$ to generate the updated factor feature $\tilde{\mathbf{g}}_c$.

Algorithm 1 The FGNN layer

```
Input: $\mathcal{G} = (\mathcal{V}, \mathcal{C}, \mathcal{E}), [\mathbf{f}_i]_{i\in\mathcal{V}}, [\mathbf{g}_c]_{c\in\mathcal{C}}, [t_{ci}]_{(c,i)\in\mathcal{E}}$
Output: $[\tilde{\mathbf{f}}_i]_{i\in\mathcal{V}}$, $[\tilde{\mathbf{g}}_c]_{c\in\mathcal{C}}$
1: Variable-to-Factor:
2:     $\tilde{\mathbf{g}}_c = \underset{i:(c,i)\in\mathcal{E}}{\mathbf{AGG}} \mathcal{Q}(\mathbf{t}_{ci} \,|\Phi_{\mathrm{VF}})\, \mathcal{M}([\mathbf{g}_c, \mathbf{f}_i]|\Theta_{\mathrm{VF}})$
3: Factor-to-Variable:
4:     $\tilde{\mathbf{f}}_i = \underset{c:(c,i)\in\mathcal{E}}{\mathbf{AGG}} \mathcal{Q}(\mathbf{t}_{ci} \,|\Phi_{\mathrm{FV}})\, \mathcal{M}([\ \mathbf{g}_c, \mathbf{f}_i]|\Theta_{\mathrm{FV}})$
```

By stacking multiple FGNN layers together, we obtain a FGNN that transforms the initial factor graph features to variable and factor embeddings. To assist learning, we can add other architecture details such as residual connections. As graph neural networks, unlike belief propagation algorithms, the parameters do not need to be tied across different layers, potentially giving better representational power. Furthermore, the latent variables can have different dimensions in different layers. Different types of layers such as fully connected layers can also be interleaved with the FGNN layers.

4.4 FGNN for Max-Product Belief Propagation

We motivated the construction of FGNN based on Low-rank Sum-Product LBP message updates. In this section, we prove that another widely used approximate inference algorithm, Max-Product Belief Propagation can be exactly parameterized by the FGNN using *maximization* as **AGG** operator in Algorithm (1). For convenience of description, we use a log-linear formulation of PGMs.

Let $\mathbf{x}$ be the set of all variables and let $\mathbf{x}_c$ be the subset of variables that factor ϕ_c depends on. Denote the set of indices of variables in $\mathbf{x}_c$ by $s(c)$. Then, an equivalent formulation of PGMs in equation (2) is as follows

$$p(\mathbf{x}) = \frac{1}{Z} \exp\left[\sum_{c\in\mathcal{C}} \theta_c(\mathbf{x}_c) + \sum_{i\in\mathcal{V}} \theta_i(x_i)\right], \tag{24}$$

where $\exp(\theta_c(\mathbf{x}_c)) = \phi_c(\mathbf{x}_c)$ and $\exp(\theta_i(x_i)) = \phi_i(x_i)$ are non-negative factor potential functions (with $\theta_c(\cdot)$, $\theta_i(\cdot)$ as the corresponding log-potentials) and Z is a normalizing constant. Using the log-linear formulation, the MAP inference problem in (4) can be reformulated as

$$\mathbf{x}^* = \operatorname*{argmax}_{\mathbf{x}} \sum_{c\in\mathcal{C}} \theta_c(\mathbf{x}_c) + \sum_{i\in\mathcal{V}} \theta_i(x_i), \tag{25}$$

and the Max-Product loopy belief propagation in Eqns 8, 9, 10 can be reformulated as

$$n_{i\to c}(x_i) = \theta_i(x_i) + \sum_{d:d\neq c, i\in s(d)} m_{d\to i}(x_i), \tag{26a}$$

$$m_{c\to i}(x_i) = \max_{\mathbf{x}_c\setminus\{x_i\}} \left[\theta_c(\mathbf{x}_c) + \sum_{j\in s(c), j\neq i} n_{j\to c}(x_j)\right], \tag{26b}$$

$$b_i(x_i) = \theta_i(x_i) + \sum_{c:i\in s(c)} m_{c\to i}(x_i) \tag{26c}$$

We prove that Max-Product Belief Propagation can be exactly parameterized by the FGNN. The proof of the propositions and lemmas are provided in the appendix. The sketch of the proof is as follows. First, instead of parameterizing higher-order potentials as sum of rank-1 tensors, we show that the higher-order potentials can be also decomposed as maximization over a set of rank-1 tensors, and that the decomposition can be represented by a FGNN layer. After the decomposition, a single Max-Product iteration only requires two operations: (1) maximization over rows or columns of a matrix, and (2) summation over a group of features. We show that the two operations can be exactly parameterized by the FGNN and that k Max-Product iterations can be simulated using $\mathcal{O}(k)$ FGNN layers.

Initially, we represented the higher-order potential functions as sum of rank-1 tensors which is suitable for Sum-Product type inference algorithms. However for Max-Product type algorithms, a decomposition as a maximum of a finite number of rank-1 tensors is more appropriate. It has been shown that there is always a finite decomposition of this type (Kohli and Kumar, 2010).

Lemma 1 ((Kohli and Kumar, 2010)). *Given an arbitrary potential function $\phi_c(\mathbf{x}_c)$, there exists a variable $z_c \in \mathcal{Z}_c = \{1, 2, \ldots, Z_c\}$ with $Z_c < \infty$ and a set of univariate potentials $\{\phi_{ic}(x_i, z_c) | i \in c\}$, s.t.*

$$\log\phi_c(\mathbf{x}_c) = \log\max_{z_c\in\mathcal{Z}_c} \prod_{i\in s(c)} \phi_{ic}(x_i, z_c) = \max_{z_c\in\mathcal{Z}_c} \sum_{i\in s(c)} \varphi_{ic}(x_i, z_c), \tag{27}$$

where $\varphi_{ic}(x_i, z_c) = \log\phi_{ic}(x_i, z_c)$.

In Lemma 1, a higher-order potential are formulated as maximization over a set of rank-one tensors. It is notable that in worst cases, the size of set $\mathcal{Z}_c$, denoted by Z_c, is exponential against the order of the factor, but in practice higher-order potentials with low-rank properties can often be decomposed as (27) with relatively small Z_c. Using ideas from (Kohli and Kumar, 2010), we show that a PGM can be converted into single layer FGNN with the non-unary potentials represented as a finite number of rank-1 tensors.

Proposition 2. *A factor graph $\mathcal{G} = (\mathcal{V}, \mathcal{C}, \mathcal{E})$ with variable log potentials $\theta_i(x_i)$ and factor log potentials $\varphi_c(\mathbf{x}_c)$ can be converted to a factor graph $\mathcal{G}'$ with the same variable potentials and the decomposed log-potentials $\varphi_{ic}(x_i, z_c)$ using a one-layer FGNN.*

With the decomposed higher-order potential, one iteration of the Max-Product (26) can be rewritten using the following three steps [1]:

$$b_{c\to i}(z_c) \leftarrow \sum_{j\in s(c), j\neq i} \max_{x_j} \left[\varphi_{jc}(x_j, z_c) - m_{c\to j}(x_j) + b_j(x_j)\right], \tag{28a}$$

$$m_{c\to i}(x_i) \leftarrow \max_{z_c} \left[b_{c\to i}(z_c) + \varphi_{ic}(x_i, z_c)\right], \tag{28b}$$

$$b_i(x_i) \leftarrow \theta_i(x_i) + \sum_{c: i\in s(c)} m_{c\to i}(x_i) \tag{28c}$$

Given the log potentials represented as a set of rank-1 tensors at each factor node, we show that each iteration of the Max-Product message passing update can be represented by a Variable-to-Factor (VF) layer and a Factor-to-Variable (FV) layer, forming a FGNN layer, followed by a linear layer (that can be absorbed into the VF layer for the next iteration).

With decomposed log-potentials, Max-Product belief propagation mainly requires two operations: (1) maximization over rows or columns of a matrix; (2) summation over a group of features. We first show that the maximization operation in (28a) and (28b) (producing max-marginals) can be done using neural networks that can be implemented by the $\mathcal{M}$ units in the VF layer.

Proposition 3. *For arbitrary feature matrix $\mathbf{X} \in \mathbb{R}^{k\times l}$ with x_{ij} as its entry in the i^{th} row and j^{th} column, the feature mapping operation $\hat{\mathbf{x}} = [\max_j x_{ij}]_{i=1}^{l}$ can be exactly parameterized with a $2\log_2 l$-layer neural network with at most $\mathcal{O}(l^2 \log_2 l)$ parameters.*

Following the maximization operations, Eq. (28a) requires summation of a group of features. However, the VF layer uses max instead of sum operators to aggregate features. Assuming that the $\mathcal{M}$ operator has performed the maximization component of equation (28a) producing max-marginals, Proposition 4 shows how the $\mathcal{Q}$ layer can be used to produce a matrix $\mathbf{W}$ that converts the max-marginals into an intermediate form to be used with the max aggregators. The output of the max aggregators can then be transformed with a linear layer ($\mathbf{Q}$ in Proposition 4) to complete the computation of the summation operation required in equation (28a). Hence, equation (28a) can be implemented using the VF layer together with a linear layer that can be absorbed in the $\mathcal{M}$ operator of the following FV layer.

1. Detailed derivation are in appendix

Proposition 4. *For arbitrary non-negative valued feature matrix $\mathbf{X} \in \mathbb{R}_{\geqslant 0}^{k \times l}$ with x_{ij} as its entry in the i^{th} row and j^{th} column, there exists a constant tensor $\mathbf{W} \in \mathbb{R}^{k \times l \times kl}$ that can be used to transform $\mathbf{X}$ into an intermediate representation $y_{ir} = \sum_{ij} x_{ij} w_{ijr}$, such that after maximization operations are done to obtain $\hat{y}_r = \max_i y_{ir}$, we can use another constant matrix $\mathbf{Q} \in \mathbb{R}^{l \times kl}$ to obtain $[\sum_i x_{ij}]_{j=1}^{l} = \mathbf{Q}[\hat{y}_r]_{r=1}^{kl}$.*

Eq. (28b) and (28c) can be implemented in the same way as (28a) by the FV layer. First the max operations are done by the $\mathcal{M}$ units to obtain max-marginals. The max-marginals are then transformed into an intermediate form using the $\mathcal{Q}$ units which are further transformed by the max aggregators. An additional linear layer is then sufficient to complete the summation operation required in (28c). The final linear layer can be absorbed into the next FGNN layer, or as an additional linear layer in the network in the case of the final Max-Product iteration.

Using the above two proposition, we can implement all important operations (28). Firstly, by Proposition 3, we can construct the Variable-to-Factor module using the following proposition.

Proposition 5. *The operation in* (28a) *can be parameterized by one MPNN layer with $\mathcal{O}(|X| \max_{c \in \mathcal{C}} |\mathcal{Z}_c|)$ parameters followed by a $\mathcal{O}(\log_2 |X|)$-layer neural network with at most $\mathcal{O}(|X|^2 \log_2 |X|)$ hidden units.*

Meanwhile, by Proposition 3 and Proposition 4 the Factor-to-Variable module can be constructed using the following proposition.

Proposition 6. *The operation in* (28c) *can be parameterized by one MPNN layer, where the $\mathcal{Q}$ network is identity mapping and the $\mathcal{M}$ network consists of a $\mathcal{O}(\max_{c \in \mathcal{C}} \log_2 |\mathcal{Z}_c|)$-layer neural network with at most $\mathcal{O}(\max_{c \in \mathcal{C}} |\mathcal{Z}_c|^2 \log_2 |\mathcal{Z}_c|)$ parameters and a linear layer with $\mathcal{O}(\max_{c \in \mathcal{C}} |c|^2 |X|^2)$ parameters.*

Using the above two proposition, we achieves the main result of this section as follows.

Corollary 7. *The Max-Product algorithm in* (26) *can be exactly parameterized by the FGNN, where the number of parameters are polynomial w.r.t $|X|$, $\max_{c \in \mathcal{C}} |\mathcal{Z}_c|$ and $\max_{c \in \mathcal{C}} |c|$.*

5. Experiments

In this section, we evaluate our FGNN over a diverse set of tasks. First, we evaluate FGNN's performance for graphical model inference. We create synthetic PGMs in both low order and higher-order settings, and compare FGNN with other PGM inference algorithms. We also conduct experiments on low-density parity check (LDPC) decoding task (a typical PGM inference task).

Next, we want to study how does FGNN perform on real-world data as against other state-of-the-art models. For this, we evaluate FGNN over three real-world problems where capturing higher-order dependencies is likely useful. We report experiments on the graph matching problem formulated as a PGM inference problem. We performed experiments on handwritten character recognition within words to demonstrate that FGNN is able to exploit sequence information. To validate the effectiveness of FGNN in capturing higher-order information from more general graph structured data, we report results on molecular datasets

Table 1: Results (percentage agreement with MAP and standard error) on synthetic datasets with runtime in microseconds in bracket (exact followed by approximate inference runtime for AD3). A brief description of the three datasets is as follows. D1: random unary potentials + fixed parameter in pairwise and higher order potentials; D2: random parameter in unary and pairwise potentials + fixed parameter in higher-order potentials; D3: random parameter in all potentials.

	AD3	Max-Product	MPLP	MPNN	Ours
D1	80.7±0.0014 (5 / 5)	57.3±0.0020 (6)	65.8±0.0071 (57)	71.9±0.0009 (131)	**92.5**±0.0012 (144)
D2	83.8±0.0014 (532 / 325)	50.5±0.0053 (1228)	68.5±0.0074 (55)	74.3±0.0009 (131)	**89.1**±0.0010 (341)
D3	88.1±0.0006 (91092 / 1059)	53.5±0.0081 (4041)	64.2±0.0056 (55)	82.1±0.0008 (121)	**93.2**±0.0006 (382)

and compare with other state-of-the-art k-order GNNs that capture higher-order information as well. Finally, we show FGNN is well suited for modeling human motion prediction task.

5.1 MAP Inference over Synthetic PGMs

We first evaluate FGNN on synthetically constructed datasets on graphical model inference tasks. As FGNN is based on inference algorithms, we test whether FGNN is able to outperform other prominent solvers on these inference problems on multiple datasets. First we describe experiments on chain-structured graphical model and then provide further results on other MRF structures.

Data We construct three synthetic datasets (D1, D2 and D3) for this experiment. All models start with a length-30 chain structure with binary-states nodes with node potentials randomly sampled from the uniform distribution over the interval $[0, 1]$, $\mathcal{U}[0, 1]$, and pairwise potentials that encourage two adjacent nodes to take state 1, *i.e.*, it gives high value to configuration $(1, 1)$ and low value to other configurations. In D1, the pairwise potentials are fixed, while in the others, they are randomly generated. For D1, D2, and D3, a budget higher-order potential (Martins et al., 2015) is added at every node; these potentials allow at most k of the 8 variables within their scope to take the state 1; specifically, we set $k = 5$ in D1 and D2 and set k randomly in D3.

In this paper, we use the simplest, but possibly most flexible method of defining factors in FGNN: we condition the factors on the input features. Specifically, for the problems in this section, all parameters that are not fixed are provided as input factor features. We test the ability of the proposed model to find the MAP solutions, and compare the results with a well known graph neural network, MPNN (Gilmer et al., 2017) as well as several MAP inference solvers, namely AD3 (Martins et al., 2015) which solves a linear programming relaxation using subgradient methods, Max-Product Belief Propagation (Weiss and Freeman, 2001), implemented by (Mooij, 2010), and a convergent version of Max-Product – MPLP (Globerson and Jaakkola, 2008), also based on a linear programming relaxation. The approximate inference algorithms are run with the correct models while the graph neural network models use learned models, trained with exact MAP solutions generated by a branch-and-bound solver that uses AD3 for bounding (Martins et al., 2015)[2].

2. When obtaining the ground truth, we let the upper bound to touch the lower bound in the branch-and-bound solver to guarantee the optimality.

Architecture and training details In this task, we use a factor graph neural network consisting of 8 FGNN layers (see the detail in the Appendix). The model is implemented in pytorch (Paszke et al., 2017), and trained with Adam optimizer (Kingma and Ba, 2014) with initial learning rate $\text{lr} = 3 \times 10^{-3}$. After each epoch, lr is decreased by multiplying a factor of 0.98. All the models in Table 1 were trained for 50 epochs after which all of them achieve convergence.

Results The percentage of agreement with MAP solutions is shown in Table 1. Our model achieves far better results on D1, D2, and D3 than all others. D4 consists of chain models, where Max-Product works optimally[3]. The linear programming relaxations also perform well. In this case, our method is able to learn a near-optimal inference algorithm on the chain case as well.

Traditional methods including Max-Product, MPLP perform poorly on D1, D2 and D3. In these, even though FGNN can emulate the traditional Max-Product, it is better to learn a different inference algorithm. AD3 have better performance than others, but is worse than our FGNN. The accuracy of FGNN is noticeably higher than that of MPNN as MPNN does not use the higher-order structural priors that are captured by FGNN.

5.1.1 Ablation studies

In order to study the behaviour of the FGNN model under varying inputs, we conducted the following additional experiments as part of the ablation study using synthetic data.

Effect of wrong graph structures First we did a small ablation study by modifying the graph structure inputed to FGNN using the Dataset D1 and D2. Originally D1 and D2 provides chain structured PGM, with budget higher-order factor formed over every 8 neighbor variables. In this experiment we augment the graph structure by using 4 and 6 variables to form a higher-order factor instead of the correct 8 variables. On D1, the accuracies are 81.7 and 89.9 when 4 and 6 variables are used in place of the correct 8 variables; on D2, the accuracies are 50.7 and 88.9 respectively. In both cases, the highest accuracies are achieved when the sizes of the HOPs are set correctly.

Generalization to novel graph structures In order to further evaluate the generalization of FGNN, we conducted an additional experiment to train the FGNN on fixed length-30 MRFs using the same protocol as Dataset3, and test the algorithm on 60000 random generated chain MRF whose length ranges from 15 to 45 (the potentials are generated using the same protocol as D3, where all unary, pairwise and higher-order potentials have random parameters). The result is in Table 2, which shows that the model trained on fixed size MRF can be generalized to MRF with different graph structures.

Effect on low order PGMs such as chains and trees In addition to the higher-order PGM above, we conducted an additional experiment on chain and tree structured PGMs. For chain structured PGMs, we use the same protocol as D3 to generate training and testing data, but with higher-order factors removed. For tree structured dataset, the training set includes 90000 different PGMs as randomly generated binary trees whose depths are between

3. Additional experiment on trees, where Max-Product also works optimally can be found in Appendix 5.1.1 along with details on all experiments.

Chain length	AD3	FGNN
(15, 25)	88.95	94.31
(25, 35)	88.18	93.64
(35, 45)	87.98	91.50

Table 2: Generalization ability (measured by percentage agreement with MAP) of algorithms on PGMs with different graph structures. We train our FGNN model on the training set of D3 where structure of PGMs is fixed and test it on higher-order PGMs with different chain-length.

3 and 6. Each node is associated with a random variable $x_i \in \{0, 1\}$ along with a log potential $\theta_i(x_i)$ randomly sampled from Gaussian distribution $\mathcal{N}(0, 1)$. Each edge (i, j) in a tree is associated with a pairwise log potential $\theta_{ij}(x_i, x_j)$ which is randomly sampled from Gaussian distribution $\mathcal{N}(0, 1)$. There is also 10000 testing instances which are generated in the same way as the training set. The experiment result is shown in Table 3.

	AD3	Max-Product	MPLP	MPNN	FGNN
Chain	100	100	99.9	91.2	98.0
Tree	100.0	100.0	99.97	–	98.35

Table 3: Results (percentage agreement with MAP) on chain and tree structured PGMs.

For chain PGMs, our algorithm achieves comparable results as Max-Product, which is known to be optimal on chain PGMs, and outperforms pairwise MPNN. For a tree structured PGM, it is not as easy to shrink the pairwise features to the nodes as an adaptation for MPNN as in the case of chain PGM in Section 5.1, so we omit the experiment on MPNN. Still, our Factor Graph Neural Network achieves a good performance even when compared with Max-Product which is optimal on tree PGMs and also with the linear programming relaxations.

5.2 LDPC Decoding (MAP or Marginal Inference)

The low-density parity check (LDPC) codes are widely used in wired and wireless communication, where the decoding can be done by Sum- or Max-Product belief propagation (Zarkeshvari and Banihashemi, 2002).

Data Let $\mathbf{x}$ be the 48-bit original signal, and $\mathbf{y}$ be the 96-bit LDPC encoded signal by encoding scheme "96.3.963"(MacKay, 2009). Then a noisy signal $\tilde{\mathbf{y}}$ is obtained by transferring $\mathbf{y}$ through a channel with white Gaussian and burst noise, that is, for each bit i, $\tilde{y}_i = y_i + n_i + p_i z_i$, where $n_i \sim \mathcal{N}(0, \sigma^2)$, $z_i \sim \mathcal{N}(0, \sigma_b^2)$, and p_i is a Bernoulli random variable *s.t.* $P(p_i = 1) = \eta$; $P(p_i = 0) = 1 - \eta$. In the experiment, we set $\eta = 0.05$ as (Kim et al., 2018) to simulate unexpected burst noise during transmission. By tuning σ, we can get different signal with $\text{SNR}_{dB} = 20 \log_{10}(1/\sigma)$.

In the experiment, for all learning-based methods, we generate $\tilde{\mathbf{y}}$ from randomly sampled $\mathbf{x}$ on the fly with $\text{SNR}_{dB} \in \{0, 1, 2, 3, 4\}$ and $\sigma_b \in \{0, 1, 2, 3, 4, 5\}$. For each learning-based method, 10^8 samples are generated for training. Meanwhile, for each different SNR_{dB} and σ_b, 1000 samples are generated for validating the performance of the trained model.

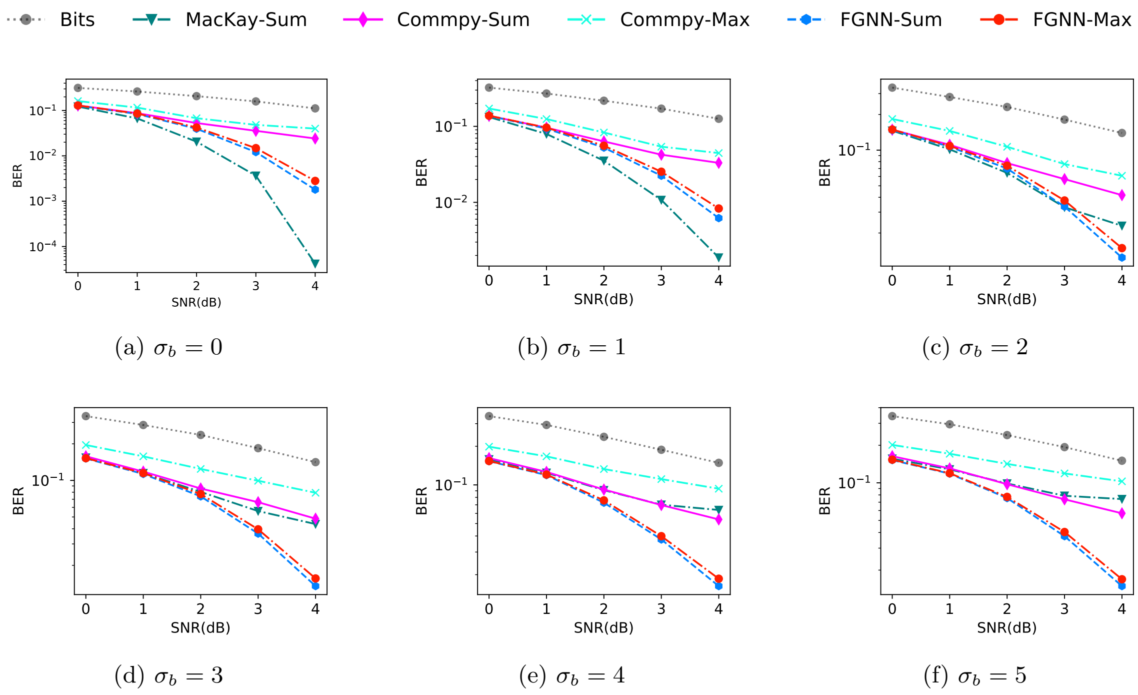


Figure 5: Experimental results (Bit Error Rate, BER v.s. Signal-to-Noise Ratio, SNR) on LDPC decoding.

In LDPC decoding, the SNR_{dB} is usually assumed to be known and fixed, and the burst noise is often unexpected and its parameters are unknown to the decoder. Thus, for learning based methods and traditional LDPC decoding method, the noisy signal $\tilde{\mathbf{y}}$ and the SNR_{dB} are provided as input. In our experiments, since the LDPC decoding can be done by both max-product and sum-product, we train our FGNN with both max and sum aggregation function (see FGNN-Max and FGNN-Sum in Figure 5.). The FGNN is compared with baselines including two implementation of Sum-Product (Mackey-Sum (MacKay, 2009) and Commpy-Sum (Taranalli, 2020)), the max-product decoder from Commpy (Commpy-Max (Taranalli, 2020)) and the bit decoding baseline.

Architecture and training details In this task, we use a factor graph neural network consisting of 8 FGNN layers (the details are provided in the Appendix). The model is implemented by using pytorch (Paszke et al., 2017), and trained with Adam optimizer (Kingma and Ba, 2014) with initial learning rate $\mathrm{lr} = 1 \times 10^{-2}$. After every 10000 samples, lr is decreased by multiplying a factor of 0.98. After training over 10^8 samples, the training loss converges.

Results We compare FGNN with three public available LDPC decoder MacKay-Sum (MacKay, 2009), Commpy-Sum (Taranalli, 2020) and Commpy-Max (Taranalli, 2020). The first two decoders are using Sum-Product belief propagation to propagate information between higher-order factors and nodes, but with different belief clipping strategy and different belief propagation scheduler. The third decoder are using Max-Product for as inference algorithm. Meanwhile, our FGNN uses a learned factor-variable information propagation scheme, and the other learning based method, MPNN ignores the higher-order dependencies. The decoding accuracy is provided in Figure 5. The Sum-Product based methods (MacKay-Sum and Commpy-Sum) are known to be near optimal for Gaussian noise, however the hyper-parameter of Sum-Product are sensitive. Due to different hyper-parameters, MacKay-Sum get the best performance when the burst noise level is low while Commpy-Sum have superior performance than MacKay-Sum with high burst noise level. Our FGNN (both FGNN-Max and FGNN-Sum) always performs better than the Commpy-Sum and Commpy-Max, it achieves comparable but lower performance than the MacKay-Sum for low burst noise level($\sigma_b \in [0, 2]$), and outperforms all other methods for high burst noise levels ($\sigma_b \in [3, 5]$), and the results from "sum" aggregation function are slightly better than the results from "max" aggregation function.

5.3 The Graph Matching Problem (MAP Inference)

The Graph Matching is a fundamental problem by itself and a key step in various computer vision topics including image registration, tracking, motion analysis and more. Traditionally, the graph matching problems are often modelled as Quadratic Assignment Problems (QAPs) and they can be also viewed as MAP inference problems over factor graphs (Zhang et al., 2016; Swoboda et al., 2017). These problems can be solved by using belief propagations in PGMs (Zhang et al., 2016; Swoboda et al., 2017), or using graph neural networks (Wang et al., 2019; Zhang and Lee, 2019). In this section, we apply our FGNN to graph matching problems and compare with both traditional methods as well as recent graph neural network based approaches.

The Problem and Traditional Model Let $\mathcal{P} = \{\mathbf{p}_i \,|i \in [n]\}$, $\mathcal{Q} = \{\mathbf{q}_i \,|i \in [n]\}$ be two sets of feature points, where $[n]$ denotes the set $\{1, 2, \dots, n\}$. The graph matching problem can be formulated as an MAP inference problem as follows

$$\operatorname*{argmax}_{\mathbf{X}} \sum_{i,j=1}^{n} x_{ij} S_{\mathrm{n}}(\mathbf{p}_i, \mathbf{p}_j) + \sum_{i,j,k,l=1}^{n} x_{ik} x_{jl} S_{\mathrm{e}}(\mathbf{e}^{\mathcal{P}}_{ij}, \mathbf{e}^{\mathcal{Q}}_{kl}) + \phi(\mathbf{X}), \tag{29}$$

where S_{n} and S_{e} are user-specified similarity functions of features, and $\mathbf{e}^{\mathcal{P}}_{ij}$ and $\mathbf{e}^{\mathcal{Q}}_{kl}$ are edge features extracted or learned from $\mathcal{P}$ and $\mathcal{Q}$, respectively. The variable $\mathbf{X}$ is a $n \times n$ matrix with x_{ij} as its entry at i^{th} row and j^{th} column. The higher-order log-potential $\phi(\mathbf{X})$ is used to enforce the one-to-one constraints as follows,

$$\phi(\mathbf{X}) = \begin{cases} 0, & \sum_{i=1}^{n} x_{ij} = 1, \sum_{j=1}^{n} x_{ij} = 1, x_{ij} \geqslant 0 \\ -\infty, & \text{otherwise.} \end{cases} \tag{30}$$

Traditionally, both the features and the similarity functions are handcrafted (Wang et al., 2018; Swoboda et al., 2017; Liu et al., 2014; Zhou and De la Torre, 2012; Zhang et al., 2016). Recently, many approaches were proposed (Wang et al., 2021; Xu et al., 2021; Rolínek et al., 2020; Fey et al., 2020; Wang et al., 2020; Yu et al., 2019; Wang et al., 2019) to replace the handcrafted feature with learned feature with very promising results.

The FGNN Model In the traditional formulation of graph matching problem (29), there are n^2 binary variables. We instead use a more compact but equivalent formulation where there are $2n$ variables with n states. We formulate the graph matching problem as

$$\underset{\mathbf{y},\mathbf{z}}{\text{argmax}} = \sum_{i=1}^{n} \theta_i(y_i | \mathcal{P}, \mathcal{Q}) + \sum_{j=1}^{n} \vartheta_j(z_j | \mathcal{P}, \mathcal{Q}) + \varphi(\mathbf{y}, \mathbf{z} | \mathcal{P}, \mathcal{Q}), \tag{31}$$

where $y_i = j$ indicates that the i^{th} node in source graph corresponding to the j^{th} node in the target graph. Similarly, $z_j = i$ indicates that the the i^{th} node in source graph corresponds the j^{th} node in the target graph. The higher-order term can enforce that $\mathbf{y}$ and $\mathbf{z}$ being consistency, and can absorb the pairwise terms in (29). As a result, we can extend the graph neural network in Zhang and Lee (2019) to handle higher-order terms more efficiently by using the proposed factor graph neural network shown in Figure 6, where each node in source graph corresponds to random variable $\mathbf{y}_i$ and each node in target graph corresponds to random variable $\mathbf{z}_i$. The pairwise message passing procedure inside the FGNN is only used to produce better node features as Zhang and Lee (2019). Without factors, the model can still work as a metric-learning based method but the factors can significantly improve the performance of GNN model.

Data Following the experimental settings of (Wang et al., 2019; Yu et al., 2019; Fey et al., 2020; Rolínek et al., 2020; Xu et al., 2021), we use the Pascal VOC dataset (Everingham et al., 2010) with the keypoints annotated by Bourdev and Malik (2009) to evaluate the performance of handcrafted and learning-based graph matching algorithms. The dataset contains 20 classes of instances (objects) with manually labeled keypoint locations, and the instances may vary in scale, view angle and illumination. For each instance, the number of inliers ranges from 6 to 23. We applied the same filtering and training/testing split (Wang et al., 2019; Yu et al., 2019; Fey et al., 2020; Rolínek et al., 2020; Xu et al., 2021), where 7020 annotated images are used for training and 1682 for testing.

Architecture and training details In our experiment, we use the same training protocol as in (Wang et al., 2019; Yu et al., 2019; Fey et al., 2020; Rolínek et al., 2020; Xu et al., 2021), where the input of the models are two sets of coordinates of key-points and two images. In our model, as in previous work the two images are first been fed to the VGG19 (Simonyan and Zisserman, 2014) net to form visual features. By using bilinear interpolating as previous work (Fey et al., 2020), we can get the visual feature vector of every node from the outputs of VGG. For each set of key-points, each key-point is connected to its k-nearest neighbors with $k = 6$. For each node, the geometric and visual features are concatenated to form the node feature. For each edge, the difference of geometric node features of the two nodes connected by the edge serves as the edge feature. Furthermore, the initial factor features are generated as follows

$$\mathbf{f} = \max_{i \in f} \mathbf{v}_i, \tag{32}$$

where $\mathbf{v}_i$ is the node feature associated with node i, and max is the entrywise maximization. The above node, edge and factor features will be fed into a FGNN network composed by three blocks. In each block, there will be a pairwise message passing module as (Zhang and Lee, 2019), and one factor message passing module as 1. Then the FGNN network, along with the VGG network, will be trained with Adam (Kingma and Ba, 2014) optimizer with learning rate 10^{-4} (10^{-6} for the VGG part). Our algorithm has been trained for 200 epochs and in each epoch we random samples 16000 image pairs from the training set.

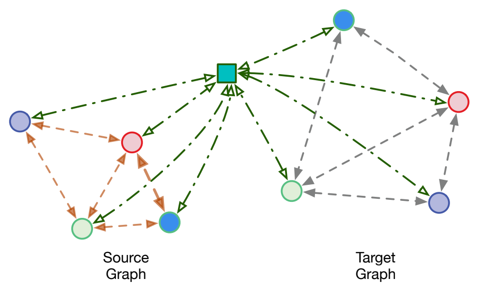


Figure 6: Factor Graph Neural Networks for Graph Matching Problems. The pairwise message passing are used to produce better node features so that without the factor it can still work as a metric-learning based methods. With the factor, the performance of matching can be significantly improved as shown in Table 4.

Method	aero	bike	bird	boat	btl	bus	car	cat	chair	cow	table	dog	horse	mbk	prn	plant	shp	sofa	trn	tv	avg
IPFP (Leordeanu et al., 2009)	25.1	26.4	41.4	50.3	43.0	32.9	37.3	32.5	33.6	28.2	26.9	26.1	29.9	32.0	28.8	62.9	28.2	45.0	69.3	33.8	36.6
RRWM (Cho et al., 2010)	30.9	40.0	46.4	54.1	52.3	35.6	47.4	37.3	36.3	34.1	28.8	35.0	39.1	36.2	39.5	67.8	38.6	49.4	70.5	41.3	43.0
PSM (Egozi et al., 2012)	32.6	37.5	49.9	53.2	47.8	34.6	50.1	35.5	37.2	36.3	23.1	32.7	42.4	37.1	38.5	62.3	41.7	54.3	72.6	40.8	43.1
GNCCP (Liu and Qiao, 2013)	28.9	37.1	46.2	53.1	48.0	36.3	45.5	34.7	36.3	34.2	25.2	35.3	39.8	39.6	40.7	61.9	37.4	50.5	67.0	34.8	41.6
ABPF (Wang et al., 2018)	30.9	40.4	47.3	54.5	50.8	35.1	46.7	36.3	40.9	38.9	16.3	34.8	39.8	39.6	39.3	63.2	37.9	50.2	70.5	41.3	42.7
GMN (Zanfir and Sminchisescu, 2018)	31.9	47.2	51.9	40.8	68.7	72.2	53.6	52.8	34.6	48.6	72.3	47.7	54.8	51.0	38.6	75.1	49.5	45.0	83.0	86.3	55.3
LCS (Wang et al., 2020)	46.9	58.0	63.6	69.9	87.8	79.8	71.8	60.3	44.8	64.3	79.4	57.5	64.4	57.6	52.4	96.1	62.9	65.8	94.4	92.0	68.5
HNN-HM(Liao et al., 2021)	39.6	55.7	60.7	76.4	87.3	86.2	77.6	54.2	50.0	60.7	78.8	51.2	55.8	60.2	52.5	96.5	58.7	68.4	96.2	92.8	68.0
PCA-GM (Wang et al., 2019)	40.9	55.0	65.8	47.9	76.9	77.9	63.5	67.4	33.7	65.5	63.6	61.3	68.9	62.8	44.9	77.5	67.4	57.5	86.7	90.9	63.8
CIE-H (Yu et al., 2019)	51.2	69.2	70.1	55.0	82.8	72.8	69.0	74.2	39.6	68.8	71.8	70.0	71.8	66.8	44.8	85.2	69.9	65.4	85.2	92.4	68.9
DGMC (Fey et al., 2020)	47.0	65.7	56.8	67.6	86.9	87.7	85.3	72.6	42.9	69.1	84.5	63.8	78.1	55.6	58.4	98.0	68.4	92.2	94.5	85.5	73.0
BBGM (Rolínek et al., 2020)	61.5	75.0	78.1	80.0	87.4	93.0	89.1	80.2	58.1	77.6	76.5	79.3	78.6	78.8	66.7	97.4	76.4	77.5	97.7	94.4	80.1
NGMv2 (Wang et al., 2021)	61.8	71.2	77.6	78.8	87.3	93.6	87.7	79.8	55.4	77.8	89.5	78.8	80.1	79.2	62.6	97.7	77.7	75.7	96.7	93.2	80.1
NHGMv2(Wang et al., 2021)	59.9	71.5	77.2	79.0	87.7	94.6	89.0	81.8	60.0	81.3	87.0	78.1	76.5	77.5	64.4	98.7	77.8	75.4	97.9	92.8	**80.4**
MPNN (Zhang and Lee, 2019)	57.8	69.1	74.4	77.7	89.2	90.4	90.4	77.4	73.1	81.9	90.4	76.5	78.6	76.5	54.4	97.9	78.2	70.0	97.3	94.9	79.8
Ours	57.3	69.0	75.9	78.3	93.8	91.6	90.8	76.9	73.9	82.5	89.9	77.0	79.9	77.8	54.4	98.2	78.2	74.9	97.7	94.7	**80.6**

Table 4: Accuracy on the Pascal VOC Keypoint dataset. **Top**: Results of traditional hand-crafted solver. **Bottom:** Result from methods using learned feature. All the learning based approaches are using VGG19 (Simonyan and Zisserman, 2014) as backbone to extract visual feature, but the graph neural network architectures are different.

Results Our FGNN based algorithm is compared with a bunch of traditional handcrafted graph matching algorithms, as well as several state-of-the-arts learned graph matching algorithms. The results are shown in Table 4, where the results of handcrafted graph matching methods are from Wang et al. (2020), and the results of learning-based approaches are from their paper except for MPNN (Zhang and Lee, 2019). For the results of MPNN, their network is identical to ours but with the factor message passing module removed, and we train the network using exactly the same protocol as ours.

Our algorithm outperforms the previous methods because our factor message passing module can handle higher-order information better. Particularly the performance of previous pairwise network based state-of-the-arts methods NGMv2 (Wang et al., 2021) can be improved by introducing higher-order terms to form NHGMv2 (Wang et al., 2021) to get an improvement of 0.3%. Meanwhile, compare to our pairwise counter-part MPNN (Zhang and Lee, 2019), we got a performance improvement of 0.8% and our average performance outperforms all previous methods.

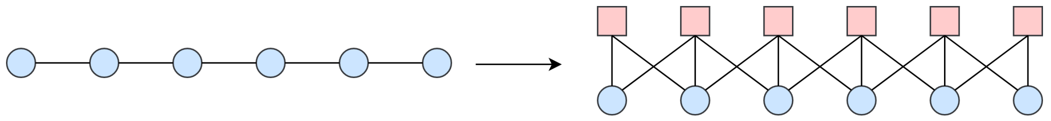


Figure 7: Factor Graph model for the Handwriting character recognition.

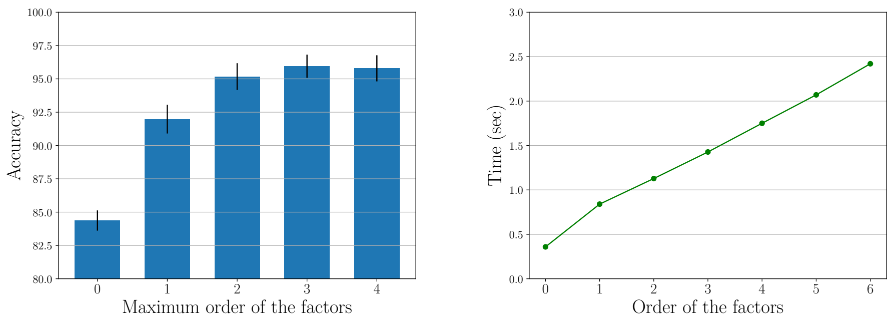


Figure 8: Handwriting character Recognition

5.4 Handwritten character sequence recognition (Marginal Inference)

In this experiment, we explore how useful higher-order modeling with FGNN is for structured prediction tasks with sequence data. A sequence is one of the simplest graph structures where nodes are connected in the form of a linear chain. FGNN should be able to capture strong higher-order dependencies in such datasets. Therefore, we explore the effect of order of factors on the task of handwritten character recognition.

Modeling In handwritten character recognition sequence, the adjacent few characters of a node are likely to contain useful information for predicting a character. We consider one of the simplest higher-order model: a $k-$order factor for each node in the sequence. Formally, let $x = [x_1, \dots, x_{|x|}]$ denote an input sequence of length $|x|$ with each x_i associated with a label $y_i \in \{a, b, \dots, z\}$. If $x_{i,k} = [x_{i-k}, \dots, x_{i+k}]$ is an order-k segment of x centered at x_i and $y_{i,k}$ is the corresponding label segment of $x_{i,k}$, we define $f_i(y_{i,k}|x_{i,k} : \theta)$ as the $k-$order factor encoding dependencies in $y_{i,k}$. This gives us a conditional random field with unnormalized probability as $P(y|x) = \prod_{i=1}^{|x|} f_i(y_{i,k}|x_{i,k}; \theta)$. This corresponds to a factor f_i for every node i connecting its adjacent k nodes in the sequence.

Data We study the properties of FGNN network with the handwriting recognition data set from (Taskar et al., 2004), originally collected by (Kassel, 1995). The dataset consists of a subset of $\sim$ 6100 handwritten words with the average length of $\sim$ 8 characters. The words are segmented into characters and each character is rasterized into an image of 16 by 8 binary pixels. The dataset is available in 10 folds with each fold containing $\sim$ 6000 characters. The task is to recognise characters by leveraging the neighbourhood of characters within the context of words. Since the words come from a small vocabulary, there is a strong higher-order correlation present in the dataset. In our framework, depending on the order, each character node x_i can share a factor with other character nodes x_j within the same word. We follow a 10-fold cross-validation setup with 1 fold for training and 9 folds for testing and report averaged results. We evaluate the performance of FGNN by varying the order and rank of the factors.

Architecture and training details We use 3 standard convolution and a fully connected layer as the base feature extractor to get the zero-th order feature of 512 dimensions. We then run 3 iterations of higher-order message passing followed by a classifier. We fix the rank of all factors at 1024 and share parameters between factors with the same order. We train for 50 epochs with a learning rate of 0.001.

Results In this experiment we study the behaviour of the model in terms of accuracy and training time when order and rank of the factors are varied. Results as shown in Figure (8) suggest the FGNN is able to capture higher-order information. The model shows strong improvements as maximum order of factors used is successively increased before saturating at 4th-order and above. To evaluate the efficiency of FGNN with higher-order factors, we analysed the computation time as we vary the order of factors used. Figure (8) shows that the training time per epoch grows almost linearly with the order of factors.

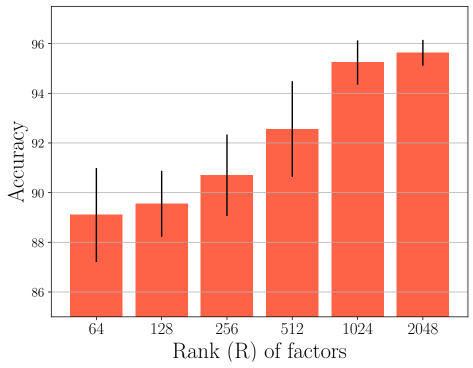


Figure 9: Effect of rank of factor parameters for Handwriting character Recognition. The accuracy of the model increases with increasing rank and saturates after rank-1024, whereas a full rank 3rd order factor would need at least 17576 components for exact representation.

Effect of Rank of factor parameters One important hyperparameter of the FGNN model is the rank of the higher-order factors which is given by the out dimension of transformation matrix $\mathbf{W}$. Since, the effect of higher-order context is clear from the results, we analyse the sensitivity of the model performance with the rank of the higher-order factors. To evaluate the effect of varying rank, we fixed the zero-th order feature dimension to 64 and used factors up to order-3. We then ran three message passing iterations by varying the rank of the factors from 64 to 2048. Results in Figure (9) show consistent improvements in performance with the increasing rank before saturating at 1024 and above. Note that for characters with the class size of 26 for each character, a full-rank tensor representation for order-3 potential function needs at least $26^3 = 17576$ components. Clearly, $1024 << 26^3$ shows that the underlying high-order potential is well approximated by the low-rank representation.

5.5 Molecular data (Graph Regression)

In the following set of experiments, we use FGNN purely as graph neural network operating on graph structured data. Note that FGNN only provides inductive bias of graphical model inference algorithms but is free to learn a richer algorithm suitable for the task. With the molecular data, we show that FGNN can work purely as a graph neural network on variable sized graphs and provides additional flexibility in modeling typed nodes and edges.

Molecular data has several properties needed for an effective study of higher-order representation learning models of graphs. A molecule can be represented as a graph with atoms as nodes and bonds that exist between atoms as edges. Higher-order information is present in the form of valence constraints of atoms which determine the number of bonds they can form. In addition, molecules often contain subgraphs, also called functional groups (*e.g.*, OH, CHO etc.), which are identifiable and significant in determining their functionality and geometry. Any relational learning model should be powerful enough to capture such higher-order structures and dependencies in order to learn highly discriminative representations. Furthermore, molecules come with varying number of nodes and hence learning higher-order functions with shareable parameters is necessary. This makes FGNN suitable for statistical learning in such datasets. We now focus on molecular data to study the effectiveness of the proposed model and show its modeling flexibility in incorporating domain knowledge in constructing the factors.

Modeling In molecular data, there is an input graph with node and edge features. To use FGNN, we need to define a suitable factor graph model based on the graph structure. In FGNN, $\mathcal{Q}(\mathbf{t}_{ci})$ is conditioned on edge features and hence separate for all variables connected to the factor. This gives much freedom in modeling to leverage domain knowledge in order to share some of the factor parameters and be able to work with large graphs with typed edges as well. Given a molecular graph, we discuss possible ways of constructing higher-order factors, conditioning and sharing of parameters.

One way to capture higher-order information is to add a factor for every node in the molecule connecting that node (we will call this node *central atom* in that factor) and its neighbours to the factor. Then weights of the potential for the factor can be shared by conditioning on the following:

- **Central atom type (CAT):** Weights within the factor are shared but different factors share parameters only if they have the same central atom type.
- **Bond type (BT):** Weights are shared if the bond type between the central atom and its neighbour is same.
- **Central atom and bond type (CABT):** Weights are shared if both the central atom type and bond type are same.
- **Central atom, bond and neighbour atom type (CABTA):** Weights are shared if the bond type and the atom types of atoms sharing the bond are same.

Do note that most molecular datasets have small number of atom types and bond types. This shows the proposed model of message passing is flexible and provides sufficient freedom in modeling.

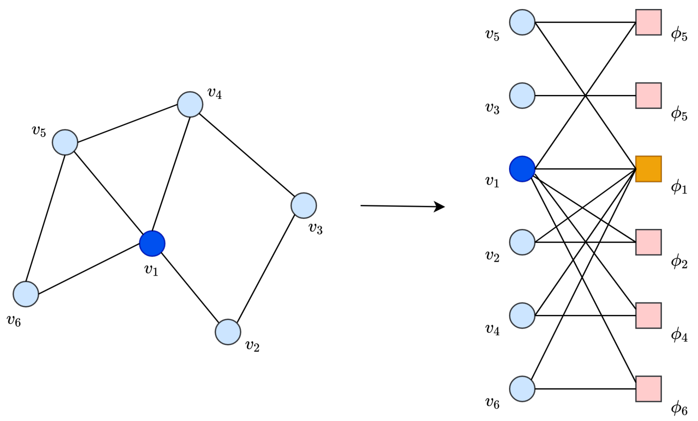


Figure 10: Factor Graph model for molecular graphs.

Data We evaluate our model on two large scale datasets on quantum chemistry, QM9 (Ruddigkeit et al., 2012; Ramakrishnan et al., 2014) and Alchemy (Chen et al., 2019) datasets on the task of regression on 12 quantum-mechanical properties. QM9 is composed of 134K drug-like organic molecules with sizes varying from 4 to 29 atoms per molecule. Atoms belong to one of the 5 types, Hydrogen (H), Carbon (C), Oxygen (O), Nitrogen(N), and Fluorine (F) and each molecule may contain up to 9 heavy (non-Hydrogen) atoms. Nodes come with discrete features along with their 3D positions. We follow the standard 80:10:10 random split for training, validation, and testing.

Alchemy (Chen et al., 2019) is a recently released more challenging dataset of 119K organic molecules comprising of 7 atom-types H, C, N, O, F, S (Sulphur) and Cl (Chlorine) with up to 14 heavy atoms. These molecules are screened for being more likely to be useful for medicinal chemistry based on the functional group and complexity. Furthermore, we follow the split based on molecule size where almost all training set contains molecules with 9 or 10 heavy atoms while the validation and test set contain molecules with 11 or 12 heavy atoms. As quantum-mechanical properties depend on the molecule size, this split tests how well the model can generalize to heavier molecules. The regression targets are the same as in QM9 dataset.

Architecture and training details We run our message passing scheme on features initialized on MPNN network with the standard implementation provided by Pytorch-geometric (Fey and Lenssen, 2019). The MPNN implementation has Edge-conv (Gilmer et al., 2017) as message function, GRU (Chung et al., 2014) as update function followed by a Set2set (Vinyals et al., 2015) function as readout for the whole graph. Readout function is required as the task is a prediction on graphs and the readout function takes in node features and outputs a vector which is used for final regression. In our implementation, we run 3 iterations of MPNN message passing scheme on input graph followed by 3 iterations of higher-order FGNN message passing described in Algorithm 1. Learning node marginals is likely to be helpful in this case as we want good node representations to be combined in the readout function. We use max function as the aggregator in VF module and summation as aggregator in FV module. We select this combination as we found it useful while being numerically stable. We then combine the MPNN output from the third iteration and the

Table 5: Graph Regression results on QM9 dataset. The error rates of MPNN, 123-GNN, PPGNN and NestedGNN are reported in different units in the respective papers. We have converted the reported numbers to units below according to multiples mentioned in Pytorch-Geometric (Fey and Lenssen, 2019). MPNN is a general GNN model and all the other baselines are learnable GNN models designed to be more expressive than MPNN. The backbone neural model of FGNN is MPNN.

		Joint training of targets					Separate training of targets				
Target	Units	MPNN	123-GNN	PPGNN	FGNN	Gain(%)	123-GNN	NestedGNN	PPGNN	FGNN	Gain(%)
μ	D	0.3580	0.4070	0.2310	**0.0920**	60.19	0.4760	0.4330	0.0934	**0.0688**	26.33
α	a_0^3	0.8900	0.3340	0.3820	**0.1830**	45.20	0.2700	0.2650	0.3180	**0.1403**	47.05
ϵ_{homo}	meV	147.21	2124	75.10	**54.15**	27.89	100.68	75.91	47.34	**45.71**	2.98
ϵ_{lumo}	meV	169.52	2310.79	78.09	**55.78**	28.57	95.51	75.10	57.14	**43.53**	22.95
Δ_ϵ	meV	179.59	2976.65	110.47	**75.91**	31.28	130.61	106.13	78.91	**65.30**	15.51
$\langle R^2 \rangle$	a_0^2	28.50	22.83	16.07	**2.81**	82.53	22.90	20.10	3.78	**1.41**	62.69
$ZPVE$	meV	58.77	307.21	17.41	**4.898**	71.87	5.170	4.08	10.85	**2.72**	33.33
U_0	meV	55783	242453	6367	**2468**	61.25	1161	5578	598	**443**	25.90
U	meV	54422	242453	6367	**2468**	61.23	3020	5442	1371	**378**	72.42
H	meV	54967	242453	6231	**2465**	60.42	1140	5775	800	**476**	40.47
G	meV	54967	242453	6476	**2468**	61.91	1276	6884	653	**364**	44.16
C_v	$\frac{\text{cal}}{\text{mol K}}$	0.42	0.1184	0.184	**0.0840**	29.05	0.0944	0.081	0.144	**0.0552**	32.09

Table 6: Graph Regression results on Alchemy dataset. The backbone neural model of FGNN is MPNN, which is also shown for comparison with FGNN.

Target	MPNN*	FGNN	Gain(%)	FGNN Network Ablation Models			
				CAT	BT	CABT	CABTA
μ	**0.1026**	0.1041	-1.41	0.1091	0.1041	0.1092	0.1233
α	0.0557	**0.0451**	19.05	0.0473	0.0451	**0.0446**	0.0449
ϵ_{homo}	0.1151	**0.1004**	12.74	0.1065	0.1004	**0.1001**	0.1007
ϵ_{lumo}	0.0817	**0.0664**	18.74	0.0712	**0.0664**	0.0685	0.0696
Δ_ϵ	0.0832	**0.0691**	16.88	0.0739	**0.0691**	0.0703	0.0720
$\langle R^2 \rangle$	0.0271	**0.0099**	63.47	0.0099	0.0099	**0.0094**	0.0120
$ZPVE$	0.0259	**0.0115**	55.42	0.0116	0.0115	**0.0108**	0.0140
U_0	0.0131	**0.0044**	65.80	**0.0042**	0.0044	0.0046	0.0054
U	0.0131	**0.0044**	65.90	**0.0041**	0.0044	0.0046	0.0054
H	0.0130	**0.0044**	65.77	**0.0042**	0.0044	0.0046	0.0054
G	0.0130	**0.0044**	65.77	**0.0042**	0.0044	0.0046	0.0054
C_v	0.0559	**0.0481**	13.93	**0.0472**	0.0481	0.0488	0.0502
MAE	0.0499	**0.0394**	21.18	0.04115	**0.0394**	0.0400	0.0424

FGNN output with concatenation followed by the set2set readout function. For FGNN module, we set the hidden vector dimension to 64. The projection dimension of VF module is set to to 512. We use Adam optimizer initialized with learning rate of $1e^{-3}$. Since we want to show improvements over MPNN, all other hyperparameters are maintained as is provided by Pytorch-geometric implementation of MPNN for a fair comparison. All targets are normalized and are trained with the absolute error loss for 200 epochs with batch size of 64.

Results For QM9 dataset, following (Maron et al., 2019) we report Mean Absolute Error (MAE) in two settings, one where all targets are jointly trained and the other where each target is separately trained. Factors are constructed as described in modeling, with factor weights conditioned on the central atom and bond type (CABT). The baselines we compare with are MPNN (Gilmer et al., 2017), 123-GNN (Morris et al., 2019), PPGNN (Maron et al., 2019) and NestedGNN (Zhang and Li, 2021). MPNN is a generalized GNN similar to that of Battaglia et al. (2018) with good performance on QM9. 123-GNN and PPGNN are the k-order methods which capture higher-order information. NestedGNN is another more expressive GNN which passes messages on rooted subgraphs instead of trees and is shown to do well on QM9 dataset. Table 5 shows that FGNN outperforms MPNN, 123-GNN, NestedGNN and PPGNN by a significant margin in all the targets under both the settings. Furthermore, the margin of improvement indicates that much of the higher-order information was not sufficiently captured by the k-order GNNs.

For Alchemy dataset, following (Chen et al., 2019) we report MAE on jointly trained normalized targets and compare with MPNN which was the best performing model in the benchmark results (Chen et al., 2019). We use the validation set to select among the models in Section 5.5. As FGNN is built on MPNN as the backend network, the margin of improvement in Table 6 is mainly because of higher-order message passing module. We also did an ablation study using the different sharing configurations. Results indicate that conditioning of parameters on either central atom or edge type helps most. Conditioning in these ways helps capture most of the higher-order information centered around an atom (node). For the CABTA method, there is a slight decrease in performance which is likely caused by the large parameter size in the model. Collectively, the ablation results suggest that major improvements are coming from the higher-order message passing scheme itself since conditioning on only bond types (BT) seems to be sufficient for a better performance.

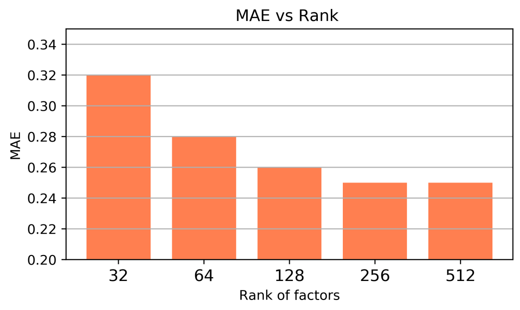


Figure 11: Effect of rank of factor parameters on QM9 dataset. Mean absolute error of the model consistently decreases with the increasing rank and saturates after rank-256. In contrast, an exact representation of the full-rank tensor of the order-5 potential would need at least 3125 components.

Effect of rank of factor parameters We now study the sensitivity of the model performance MAE with the variation in the rank of the factor parameters on the molecular data as well. For this, we fix all the model hyperparameters described in architecture details and only vary the factor rank. Figure 11 shows the variation of MAE with the increasing factor rank on the QM9 dataset. Clearly, there is improvement in model performance with the increasing rank and saturates above 256. The rank of the higher-order factors in this task is considerably lower compared to the rank in the character recognition ablation

Table 7: Comparison of FGNN with position coordinate factors with recent baselines. The backbone model of FGNN is MPNN. All the baselines compared are more powerful models than MPNN specifically designed for exploiting 3D geometric information of molecules.

Target	Units	Joint training of targets		Seperate training of targets					
		Dimenet	FGNN	Dimenet	MGCN	EGNN	ALIGNN	SphereNet	FGNN
μ	D	0.0775	**0.0549**	0.0286	0.056	0.029	**0.0146**	0.0245	0.0373
α	a_0^3	**0.0616**	0.1364	0.0469	**0.030**	0.071	0.0561	0.0449	0.0868
ϵ_{HOMO}	meV	45.1	**38.7**	27.8	42.1	29	**21.4**	22.8	33.68
ϵ_{LUMO}	meV	41.1	**37.4**	19.7	57.4	25	19.5	**18.9**	31.26
$\Delta\epsilon$	meV	59.2	**53.7**	34.8	64.2	48	38.1	**31.1**	48.7
$\langle R^2 \rangle$	a_0^2	**0.345**	1.40	0.331	0.11	**0.106**	0.543	0.268	0.365
ZPVE	meV	**2.87**	3.51	1.29	**1.12**	1.55	3.1	**1.12**	1.8
U_0	meV	**12.9**	36.3	8.02	12.9	11	15.3	**6.26**	21.5
U	meV	**13.0**	36.6	7.89	14.4	12	14.4	**6.36**	20.6
H	meV	**13.0**	36.7	8.11	16.2	12	14.7	**6.33**	27.4
G	meV	**13.8**	36.8	8.98	14.6	12	14.4	**7.78**	26.9
c_v	$\frac{\text{cal}}{\text{mol K}}$	**0.0309**	0.057	0.0249	0.038	0.031	NA	**0.0215**	0.0249

in Figure 9. This can be explained with the different class sizes of the variables in both the tasks. The variables in the molecular data belong to 5 atom types only, whereas the variables in character-recognition had a class size of 26. Considering a factor of order-5 (for Carbon atom), a full-rank tensor representation for the potential function would need at least $5^5 = 3125$ components for exact representation. Both of our ablation experiments on handwriting recognition and molecular data provide consistent evidence that the higher-order factors can be approximated with a mixture of relatively small number of rank-1 tensors.

5.5.1 QM9 with positional information

The QM9 dataset comes with additional atom coordinate features which can provide the positional as well as directional information. These directional features have been shown to be useful in Directional message passing network (Dimenet) (Klicpera et al., 2020), which was further confirmed in subsequent works of ALIGNN (Choudhary and DeCost, 2021), SphereNet (Liu et al., 2022) and GEM (Fang et al., 2022). The directional features, *i.e.* positional and angular information, used in Dimenet are known to help substantially in QM9 dataset. Specifically, Dimenet extracts the features from triplets of nodes with each triplet feature constructed from the angular information within the triplet encapsulated within a basis function. In order to incorporate this informative directional information, we augment our FGNN model with the additional factors along with node positional coordinates. We conduct additional experiments with the added factors and compare with the Dimenet and other recent baselines in performance.

Modeling and Architecture We augment the above FGNN model for molecular graphs with edge factors with positional coordinates. The edge factor connects two adjacent nodes in the molecular graph with elementwise multiplication as the aggregator function.

$$\tilde{\mathbf{g}}_{ij} = \mathcal{M}(\mathbf{f}_i \,|\Theta_{\text{VF}_i}) \odot \mathcal{M}(\mathbf{f}_j \,|\Theta_{\text{VF}_j}) \tag{33}$$

where $\mathbf{f}_i$ is the node position coordinate feature and $\mathcal{M}()$ is an MLP. Note that unlike Dimenet, our model does not include features for each triplet of nodes.

Table 8: Long-term prediction error (the smaller the better) of joint angles (top) and 3D joint positions (bottom) on H3.6M. Our model share the same backbone as Mao et al. (2019), and we replace the last two layer of GNN in Mao et al. (2019) with our FGNN model.

	Walk		Eating		Smoking		Discussion		Average	
milliseconds	560	1000	560	1000	560	1000	560	1000	560	1000
convSeq2Seq(Li et al., 2018)	N/A	0.92	N/A	1.24	N/A	1.62	N/A	1.86	N/A	1.41
GNN(Mao et al., 2019)	**0.65**	0.67	0.76	**1.12**	0.87	1.57	**1.33**	1.70	0.90	1.27
DMGNN (Li et al., 2020)	0.66	0.75	**0.14**	1.14	**0.83**	**1.52**	**1.33**	**1.45**	**0.89**	**1.21**
Ours	0.67	**0.70**	0.76	**1.12**	0.88	1.57	1.35	1.70	0.91	1.27
convSeq2Seq(Li et al., 2018)	69.2	81.5	71.8	91.4	50.3	85.2	101.0	143.0	73.1	100.3
GNN(Mao et al., 2019)	55.0	60.8	68.1	79.5	42.2	70.6	93.8	119.7	64.8	82.6
Hist-Attention (Mao et al., 2020)	47.4	58.1	**50.0**	75.7	47.6	69.5	**86.6**	119.8	57.9	80.7
Ours	**44.1**	**53.5**	59.5	**73.0**	**33.0**	**61.9**	86.9	**113.5**	**55.9**	**75.5**

Results Table 7 shows the results of the augmented FGNN model on QM9 dataset in both the settings *i.e.* joint training of 12 targets and separate training of each target. Results suggest that FGNN performs competitively in most targets and under the setting of joint training of targets, FGNN is able to beat the Dimenet in four of the twelve targets. Note that FGNN uses only node position features without explicit angular features encapsulated in basis functions as in DImenet. Furthermore, for completeness, we compare with other models including MGCN (Shui and Karypis, 2020) and more recent baselines EGNN (Satorras et al., 2021), ALIGNN(Choudhary and DeCost, 2021), SphereNet(Liu et al., 2022) which all use the 3D coordinate positional information. Furthermore, note that these models, including Dimenet, are specifically designed for molecular graph structures with domain-specific inductive biases while FGNN is a general model which can be used over any graph structure. Hence, FGNN is not able to outperform these models, however, can perform competitively.

5.6 Human Motion Prediction (Sequential Prediction)

The human motion prediction aims at predicting the future motion of a human given a history motion sequence. As there are obviously higher order dependencies between joints, the factor graph neural network may help to improve the performance of the predictor. In this section, we consider the human motion prediction problem for the skeleton data, where the angle and 3d position of each joint are predicted. We build a factor graph neural network model for the skeleton data and compare the FGNN model with the state-of-the-art model based on GNN.

Data and Modeling For human motion prediction, we are using the Human 3.6M (H3.6M) dataset. In this experiment, we replace the last two GNN layer in (Mao et al., 2019)'s model with FGNN layer with the same number of output channels. The H3.6M dataset includes seven actors performing 15 varied activities such as walking, smoking *etc.*. The poses of the actors are represented as an exponential map of joints, and a special pre-processing of global translation and rotation. In our experiments, as in previous work(Li et al., 2018; Mao et al., 2019), we only predict the exponential map of joints. That is, for each joints, we need to predict a 3-dimensional feature vector. Thus we add a factor for the

3 variable for each joint [4]. Also for two adjacent joint, a factor of 6 variables are created. The factor node feature are created by put all its variable node feature together. For the edge feature, we simply use one hot vector to represent different factor-to-variable edge. For evaluation, we compared 4 commonly used action — walk, eating, smoking and discussion. The result of GNN and convSeq2Seq are taken from (Mao et al., 2019), and our FGNN model also strictly followed the training protocol of (Mao et al., 2019).

Architecture and training details We train our model on the Human3.6M dataset using the standard training-val-test split as previous works (Mao et al., 2019; Li et al., 2018; Martinez et al., 2017), and we train and evaluate our model using the same protocol as (Mao et al., 2019) (For details, see the Appendix).

Results The results are provided in Table 8. For angle error, our FGNN model achieves similar results compared to the previous state-of-the-art GNN-based method (Mao et al., 2019; Li et al., 2020), while for 3D position error, our model achieves superior performance than the state-of-the-art models (Mao et al., 2019, 2020). Furthermore, since the backbone of our model is same as that of Mao et al. (2019), the performs improvement suggests that the higher-order modeling with FGNN is able to capture better higher-order structural priors compared to the pairwise GNN.

6. Conclusion

In this paper, we derive an efficient Low-rank Sum-Product Loopy Belief Propagation procedure for inference in factor graphs. The derived update functions are *simple*—need only matrix multiplication and Hadamard product operations, and *efficient*—the complexity of message updates grows linearly with the number of variables in the factor. In order to learn better node representations with end-to-end training, we neuralize the message passing updates to give Factor Graph Neural Network (FGNN) allowing the network to capture higher-order dependencies among the variables. We then showed FGNN can also represent the execution of the Max-Product inference algorithm on probabilistic graphical models, providing a graph neural network architecture that can represent both the Sum and Max-Product belief propagation inference algorithms.

Furthermore, we showed multiple ways of modeling higher-order factors with graph structured input data. This gives us a fairly simple, flexible, powerful, and efficient message passing scheme for representation learning of graph data where higher-order information is present. We evaluated the proposed model with extensive experiments on various tasks and domains including inference in PGMs, molecular and vision datasets where it either outperforms other state-of-the-art models substantially or is at least competitive enough. The FGNN provides a convenient method of capturing arbitrary dependencies in graphs and hypergraphs, including those with typed or conditioned nodes and edges, opening up new opportunities for adding higher-order inductive biases into learning and inference problems. More importantly, it provides a deeper theoretical understanding of the relationship between graph neural networks and inference algorithms on graphical models.

4. In practice, those angles with very small variance are ignored, and these variables are not added to the factor graph

Acknowledgments

This work was supported by the National Research Foundation Singapore under its AI Singapore Program (Award Number: AISGRP- 2018-006). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore. Zhen Zhang's participation was partially supported by the Australian Research Council Grant DP160100703. Zhen Zhang and Javen Shi's participation were partially supported by Centre for Augmented Reasoning at the Australian Institute for Machine Learning.

Appendix A. Proof of propositions

A.1 Propositions for decomposing higher-order potentials

First we provide Lemma 8, which will be used in the proof of Proposition 2 and 4.

Lemma 8. *Given l non-negative feature vectors $\mathbf{f}_i = [f_{i0}, f_{i1}, \ldots, f_{ik}]$, where $i = 1, \ldots, l$, there exists l matrices $\mathbf{Q}_i$ with shape $lk \times k$ and l vector $\hat{\mathbf{f}}_i = \mathbf{Q}_i \mathbf{f}_i^T$, s.t.*

$$, \qquad [\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_l] = [\max_i \hat{f}_{i0}, \max_i \hat{f}_{i1}, \ldots, \max_i \hat{f}_{i,kl}].$$

Proof Let

$$\mathbf{Q}_i = \left[\underbrace{\mathbf{0}^{k\times k}, \ldots, \mathbf{0}^{k\times k}}_{i-1 \text{ matrices}}, \mathbf{I}, \underbrace{\mathbf{0}^{k\times k}, \ldots, \mathbf{0}^{k\times k}}_{n-i \text{ matrices}} \right]^\top, \tag{34}$$

then we have that

$$\hat{\mathbf{f}}_i = \mathbf{Q}_i \mathbf{f}_i^T = \left[\underbrace{0, \ldots, 0}_{(i-1)k \text{ zeros}}, f_{i0}, f_{i1}, \ldots, f_{ik}, \underbrace{0, \ldots, 0}_{(n-i)k \text{ zeros}} \right]^\top.$$

By the fact that all feature vectors are non-negative, obviously we have that $[\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_l] = [\max_i \hat{f}_{i0}, \max_i \hat{f}_{i1}, \ldots, \max_i \hat{f}_{i,kl}]$. ■

Lemma (8) suggests that for a group of feature vectors, we can use the $\mathcal{Q}$ operator to produce several $\mathbf{Q}$ matrices to map different vector to different sub-spaces of a high-dimensional spaces, and then our maximization aggregation can sufficiently gather information from the feature groups.

Proposition 2. *A factor graph $\mathcal{G} = (\mathcal{V}, \mathcal{C}, \mathcal{E})$ with variable log potentials $\theta_i(x_i)$ and factor log potentials $\varphi_c(\mathbf{x}_c)$ can be converted to a factor graph $\mathcal{G}'$ with the same variable potentials and the decomposed log-potentials $\varphi_{ic}(x_i, z_c)$ using a one-layer FGNN.*

Proof Without loss of generality, we assume that $\log \phi_c(\mathbf{x}_c) \geqslant 1$. Then let

$$\theta_{ic}(x_i, z_c) = \begin{cases} \frac{1}{|s(c)|} \log \phi_c(\mathbf{x}_c^{z_c}), & \text{if } \hat{x}_i = x_i^{z_c}, \\ -c_{x_i, z_c}, & \text{otherwise}, \end{cases} \tag{35}$$

where c_{x_i,z_c} can be arbitrary real number which is larger than $\max_{\mathbf{x}_c} \theta_c(\mathbf{x}_c)$. Obviously we will have

$$\log \phi_c(\mathbf{x}_c) = \max_{z_c} \sum_{i \in s(c)} \theta_{ic}(x_i, z_c) \tag{36}$$

Assume that we have a factor $c = 1, 2, \ldots n$, and each nodes can take $|X|$ states. Then $\mathbf{x}_c$ can be sorted as

$$\begin{aligned} [\,&\mathbf{x}_c^0 = [x_1 = 0, x_2 = 0, \ldots, x_n = 0], \\ &\mathbf{x}_c^1 = [x_1 = 1, x_2 = 0, \ldots, x_n = 0], \\ &\ldots, \\ &\mathbf{x}_c^{|X|^n - 1} = [x_1 = |X|, x_2 = |X|, \ldots, x_n = |X|]], \end{aligned}$$

and higher-order potential can be organized as vector $\mathbf{g}_c = [\log \phi_c(\mathbf{x}_c^0), \log \phi_c(\mathbf{x}_c^1), \ldots, \log \phi_c(\mathbf{x}_c^{|X|^n-1})]$. Then for each i the item $\theta_{ic}(x_i, z_c)$ in (35) have $|X|^{n+1}$ entries, and each entry is either a scaled entry of the vector $\mathbf{g}_c$ or arbitrary negative number less than $\max_{\mathbf{x}_c} \theta_c(\mathbf{x}_c)$.

Thus if we organize $\theta_{ic}(x_i, z_c)$ as a length-$|X|^{n+1}$ vector $\mathbf{f}_{ic}$, then we define a $|X|^{n+1} \times |X|^n$ matrix $\mathbf{Q}_{ci}$, where if and only if the l^{th} entry of $\mathbf{f}_{ic}$ is set to the m^{th} entry of $\mathbf{g}_c$ multiplied by $1/|s(c)|$, the entry of $\mathbf{Q}_{ci}$ in l^{th} row, m^{th} column will be set to $1/|s(c)|$; all the other entries of $\mathbf{Q}_{ci}$ is set to some negative number smaller than $-\max_{\mathbf{x}_c} \theta_c(\mathbf{x}_c)$. Due to the assumption that $\log \phi_c(\mathbf{x}_c) \geqslant 1$, the matrix multiplication $\mathbf{Q}_{ci}\, \mathbf{g}_c$ must produce a legal $\theta_{ic}(x_i, z_c)$.

If we directly define a $\mathcal{Q}$-network which produces the above matrices $\mathbf{Q}_{ci}$, then in the aggregating part of our network there might be information loss. However, by Lemma 8 there must exists a group of $\tilde{\mathbf{Q}}_{ci}$ such that the maximization aggregation over features $\tilde{\mathbf{Q}}_{ci}\, \mathbf{Q}_{ci}\, \mathbf{g}_c$ will produce exactly a vector representation of $\theta_{ic}(x_i, z_c), i \in s(c)$. Thus if every t_{ci} is a different one-hot vector, we can easily using one single linear layer $\mathcal{Q}$-network to produce all $\tilde{\mathbf{Q}}_{ci}\, \mathbf{Q}_{ci}$, and with a $\mathcal{M}$-network which always output factor feature, we are able to output a vector representation of $\theta_{ic}(x_i, z_c), i \in s(c)$ at each factor node c. ■

A.2 Derivation of decomposed max-product belief propagation

In this section, we reformulate the (26) using the decomposed higher-order-log potentials. We use $m_{c\to i}(x_i)$ and $b_i(x_i)$ for the previous messages and beliefs, and use $m'_{c\to i}(x_i)$ and $b'_i(x_i)$ for the updated messages and beliefs. Then message updating step in the max product belief propagation (26) can be reformulated as

$$\begin{aligned}
n_{i\to c}(x_i) =& \theta_i(x_i) + \sum_{d: d\neq c, i\in s(d)} m_{d\to i}(x_i), \\
=& \theta_i(x_i) + \sum_{d: i\in s(d)} m_{d\to i}(x_i) - m_{c\to i}(x_i) \\
=& b_i(x_i) - m_{c\to i}(x_i) \qquad \text{(37a)} \\
m'_{c\to i}(x_i) =& \max_{\mathbf{x}_c \setminus x_i} \left[\theta_c(\mathbf{x}_c) + \sum_{j\in s(c), j\neq i} n_{j\to c}(x_j) \right] \\
=& \max_{\mathbf{x}_c \setminus x_i} \left\{ \max_{z_c} \left[\sum_{j\in s(c), j\neq i} \varphi_{jc}(x_j, z_c) + \varphi_{ic}(x_i, z_c) \right] + \sum_{j\in s(c), j\neq i} n_{j\to c}(x_j) \right\} \\
=& \max_{\mathbf{x}_c \setminus x_i} \left\{ \max_{z_c} \left[\sum_{j\in s(c), j\neq i} \varphi_{jc}(x_j, z_c) + \varphi_{ic}(x_i, z_c) \right] + \sum_{j\in s(c, j\neq i)} [b_j(x_j) - m_{c\to j}(x_j)] \right\} \\
=& \max_{z_c} \max_{\mathbf{x}_c \setminus x_i} \left\{ \left[\sum_{j\in s(c), j\neq i} \varphi_{jc}(x_j, z_c) + \varphi_{ic}(x_i, z_c) \right] + \sum_{j\in s(c, j\neq i)} [b_j(x_j) - m_{c\to j}(x_j)] \right\} \\
=& \max_{z_c} \left\{ \sum_{j\in s(c), j\neq i} \max_{x_j} [\varphi_{jc}(x_j) - m_{c\to j}(x_j) + b_j(x_j)] + \varphi_{ic}(x_i, z_c) \right\} \qquad \text{(37b)}
\end{aligned}$$

Here for simplying the notation we define

$$b_{c\to i}(z_c) = \sum_{j\in s(c), j\neq i} \max_{x_j} \left[\varphi_{jc}(x_j, z_c) - m_{c\to j}(x_j) + b_j(x_j)\right], \forall c, i \in s(c)$$

and then the updating rule for beliefs can be reformulated as

$$\begin{aligned} b'_i(x_i) &= \theta_i(x_i) + \sum_{c:i\in s(c)} m'_{c\to i}(x_i) \\ &= \theta_i(x_i) + \sum_{c:i\in s(c)} \max_{z_c} \left[b_{c\to i}(z_c) + \varphi_{ic}(x_i, z_c)\right]. \end{aligned}$$

Thus finally the max-product updating rules are

$$\begin{aligned} b_{c\to i}(z_c) &\leftarrow \sum_{j\in s(c), j\neq i} \max_{x_j} \left[\varphi_{jc}(x_j, z_c) - m_{c\to j}(x_j) + b_j(x_j)\right], \\ m_{c\to i}(x_i) &\leftarrow \max_{z_c} \left[b_{c\to i}(z_c) + \varphi_{ic}(x_i, z_c)\right], \\ b_i(x_i) &\leftarrow \theta_i(x_i) + \sum_{c:i\in s(c)} m_{c\to i}(x_i) \end{aligned}$$

A.3 Recovering decomposed max-product belief propagation using FGNN

Given the log potentials represented as a set of rank-1 tensors at each factor node, we need to show that each iteration of the Max Product message passing update can be represented by a Variable-to-Factor layer followed by a Factor-to-Variable layer (forming a FGNN layer). We reproduce the update equations here.

$$b_{c\to i}(z_c) \leftarrow \sum_{j\in s(c), j\neq i} \max_{x_j} \left[\varphi_{jc}(x_j, z_c) - m_{c\to j}(x_j) + b_j(x_j)\right], \tag{38a}$$

$$m_{c\to j(x_i)} \leftarrow \max_{z_c} \left[b_{c\to i}(z_c) + \varphi_{ic}(x_i, z_c)\right], \quad b_i(x_i) \leftarrow \theta_i(x_i) + \sum_{c:i\in s(c)} m_{c\to j(x_i)} \tag{38b}$$

In the Max-Product updating procedure, we should keep all the decomposed $\varphi_{jc}(x_j, z_c) = \log \phi_{jc}(x_j, z_c)$ and all the unary potential $\theta_i(x_i)$ for use at the next layer. That requires the FGNN to have the ability to fit the identity mapping. Consider letting the $\mathcal{Q}$ network to always output identity matrix, $\mathcal{M}([\mathbf{g}_c, f_i]|\Theta_{\mathrm{VF}})$ to always output $\mathbf{g}_c$, and $\mathcal{M}([\mathbf{g}_c, f_i]|\Theta_{\mathrm{FV}})$ to always output f_i. Then the FGNN will be an identity mapping. As $\mathcal{Q}$ always output a matrix and $\mathcal{M}$ output a vector, we can use part of their blocks as the identity mapping to keep $\log \phi_{jc}(x_j, z_c)$ and $\theta_i(x_i)$. The other blocks are used to updating $b_{c\to i}(z_c)$, messages $m_{c\to j}(x_j)$, and $b_i(x_i)$.

First we show that $\mathcal{M}$ operators in the Variable-to-Factor layer can be used to construct the computational graph for the max-marginal operations.

Proposition 3. *For arbitrary real valued feature matrix $\mathbf{X} \in \mathbb{R}^{k\times l}$ with x_{ij} as its entry in the i^{th} row and j^{th} column, the feature mapping operation $\hat{\mathbf{x}} = [\max_j x_{ij}]_{i=1}^k$ can be exactly parameterized with a $2\log_2 l$-layer neural network with RELU as activation function and at most $2n$ hidden units.*

Proof Without loss of generality we assume that $k = 1$, and then we use x_i to denote x_{1i}. When $l = 2$, it is obvious that

$$\max(x_1, x_2) = \mathbf{Relu}(x_1 - x_2) + x_2 = \mathbf{Relu}(x_1 - x_2) + \mathbf{Relu}(x_2) - \mathbf{Relu}(-x_2)$$

and the maximization can be parameterized by a two layer neural network with 3 hidden units, which satisfied the proposition.

Assume that when $l = 2^a l$ for some integer $a >= 1$, the proposition is satisfied [5]. Then for $l = 2^{a+1}$, we can find $\max(x_1, \ldots, x_{2^a})$ and $\max(x_{2^a+1}, \ldots, x_{2^{a+1}})$ using two network with $2a$ layers and at most 2^{a+1} hidden units. Stacking the two neural network together would results in a network with $2i$ layers and at most 2^{i+2} parameters. Then we can add another 2 layer network with 3 hidden units to find $\max(\max(x_1, \ldots, x_{2^a}), \max(x_{2^a+1}, \ldots, x_{2^{a+1}}))$. Thus by mathematical induction the proposition is proved. ■

The update equations contain summations of columns of a matrix after the max-marginal operations. However, the VF and FV layers use max operators to aggregate features produced by $\mathcal{M}$ and $\mathcal{Q}$ operator. Assume that the $\mathcal{M}$ operator has produced the max-marginals, then we use the $\mathcal{Q}$ to produce several weight matrix. The max-marginals are multiplied by the weight matrices to produce new feature vectors, and the maximization aggregating function are used to aggregating information from the new feature vectors. We use the following propagation to show that the summations of max-marginals can be implemented by one MPNN layer plus one linear layer. Thus we can use the VF layer plus a linear layer to produce $b_{c\to i}(z_c)$ and use the FV layer plus another linear layer to produce $b_i(x_i)$. Hence to do k iterations of Max Product, we need k FGNN layers followed by a linear layer.

Proposition 4. *For arbitrary non-negative valued feature matrix $\mathbf{X} \in \mathbb{R}_{\geqslant 0}^{k\times l}$ with x_{ij} as its entry in the i^{th} row and j^{th} column, there exists a constant tensor $\mathbf{W} \in \mathbb{R}^{k\times l\times kl}$ that can be used to transform $\mathbf{X}$ into an intermediate representation $y_{ir} = \sum_{ij} x_{ij} w_{ijr}$, such that after maximization operations are done to obtain $\hat{y}_r = \max_i y_{ir}$, we can use another constant matrix $\mathbf{Q} \in \mathbb{R}^{l\times kl}$ to obtain*

$$[\sum_i x_{ij}]_{j=1}^{l} = \mathbf{Q}[\hat{y}_r]_{r=1}^{kl}. \tag{39}$$

Proof The proposition is a simple corollary of Lemma 8. The tensor $\mathbf{W}$ serves as the same role as the matrices $\mathbf{Q}_i$ in Lemma 8, which can convert the feature matrix $\mathbf{X}$ as a vector, then a simple linear operator can be used to produce the sum of rows of $\mathbf{X}$, which completes the proof. ■

In Lemma 8 and Proposition 4, only non-negative features are considered, while in log-potentials, there can be negative entries. However, for the MAP inference problem in (25), the transformation as follows would make the log-potentials non-negative without changing the final MAP assignment,

$$\tilde{\theta}_i(x_i) = \theta_i(x_i) - \min_{x_i} \theta_i(x_i), \qquad \tilde{\theta}_c(\mathbf{x}_c) = \theta_c(\mathbf{x}_c) - \min_{\mathbf{x}_c} \theta_c(\mathbf{x}_c). \tag{40}$$

5. For any l we can always padding the vector to get an $l' = 2^a$ for some interger a

As a result, for arbitary PGM we can first apply the above transformation to make the log-potentials non-negative, and then our FGNN can exactly do Max-Product Belief Propagation on the transformed non-negative log-potentials.

A.4 A Factor Graph Neural Network Module Recovering the Belief Propagation

In this section, we give the proofs of Proposition 5 and 6 by constructing two FGNN layers which exactly recover the belief propagation operation. As lower order factors can always shrank by higher-order factors, we will construct the FGNN layers on an factor graph $\mathcal{H} = (\mathcal{V}, \mathcal{F}, \hat{\mathcal{E}})$, which satisfies the following condition

1. $\forall i \in \mathcal{V}$, the associated $\theta_i(x_i)$ satisfies that $\theta_i(x_i) > 0 \forall x_i \in X$;
2. $\forall f_1, f_2 \in \mathcal{F}$, $|f_1| = |f_2|$;
3. $\forall f \in \mathcal{F}$, the corresponding $\varphi_f(\mathbf{x}_f)$ can be decomposed as

$$\varphi_f(\mathbf{x}_f) = \max_{z_f \in \mathcal{Z}} \sum_{i \in f} \varphi_{fi}(x_i, z_f), \tag{41}$$

   and $\forall i \in f, \varphi_{fi}(x_i, z_f)$ satisfies that $\varphi_{fi}(x_i, z_f) > 0$.

On factor graph $\mathcal{H}$, we construct a FGNN layer on the directed bipartite graph in Figure 12.

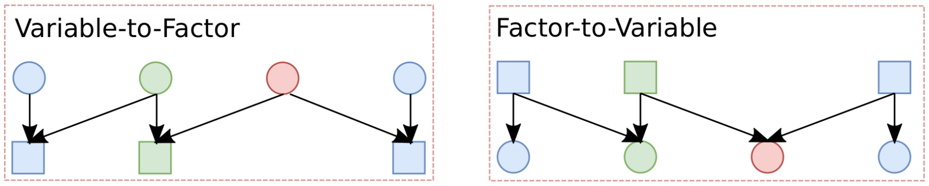


Figure 12: Directed bipartite graph for constructing FGNN layers. In the Variable-to-Factor sub-graph, each factor receives the messages from the same number of nodes. On the other hand, for each Factor-to-Variable sub-graph, each nodes may receives messages from different number of factors.

FGNN Layer to recover (28a) Here we construct an FGNN layer to produce all $b_{f\to i}(z_f)$. First we reformulate (28a) as

$$\begin{aligned} b_{f\to i}(z_f) &\leftarrow \tilde{\varphi}_f(z_f) - \max_{x_i}[\varphi_{if}(x_i, z_f) - m_{f\to i}(x_i) + b_i(x_i)], \\ \tilde{\varphi}_f(z_f) &\leftarrow \sum_{i \in f} \max_{x_i}[\varphi_{if}(x_i, z_f) - m_{f\to i}(x_i) + b_i(x_i)]. \end{aligned} \tag{42}$$

Here we use the Variable-to-Factor sub-graph to implement (42). For each variable node i, we associated it with an length-$|X|$ vector $[b_i(x_i)]_{x\in X}$ (Initially $b_i(x_i) = \theta_i(x_i)$). For each edge in the sub-graph, assume that $f = [i_1, i_2, \ldots, i_{|f|}]$, then for some $i_j \in f$, the associated feature vector is as length-$|f|$ one-hot vector as follows

$$[0, 0, \ldots, \underbrace{1}_{\text{The } j^{\text{th}} \text{ entry.}}, \ldots, 0].$$

For each factor node $f = [i_1, i_2, \dots, i_{|f|}]$ in the sub-graph, it is associated with an $|f| \times |X||Z|$ feature matrix as follows

$$\begin{bmatrix} [\varphi_{fi}(x_{i_1}, z_f) - m_{f\to i}(x_i)]_{x_{i_1}=1, z_f=1}^{x_{i_1}=|X|, z_f=|Z|} \\ [\varphi_{fi}(x_{i_2}, z_f) - m_{f\to i}(x_i)]_{x_{i_2}=1, z_f=1}^{x_{i_2}=|X|, z_f=|Z|} \\ \cdots \\ [\varphi_{fi}(x_{i_{|f|}}, z_f) - m_{f\to i}(x_i)]_{x_{i_{|f|}}=1, z_f=1}^{x_{i_{|f|}}=|X|, z_f=|Z|} \end{bmatrix}.$$

Then we construct an MPNN

$$\tilde{\mathbf{f}}_i = \max_{i \in f} \mathcal{Q}(\mathbf{e}_{f\to i})\, \mathcal{M}(\mathbf{f}_i, \mathbf{f}_f), \tag{43}$$

as follows. The $\mathcal{Q}(\mathbf{e}_{f\to i})$ is an identity mapping. The $\mathcal{M}(\mathbf{f}_i, \mathbf{f}_f)$ consists of $|f|$ addition networks, where the i_j^{th} networks will have an $|f| \times |X||Z|$ parameter

$$\begin{bmatrix} -\infty \\ -\infty \\ \cdots \\ [\varphi_{fi}(x_{i_j}, z_f) - m_{f\to i}(x_i)]_{x_{i_j}=1, z_f=1}^{x_{i_j}=|X|, z_f=|Z|} \\ \cdots \\ -\infty \end{bmatrix}.$$

In the $\mathcal{M}$-network, the $|f| \times |X||Z|$ parameter will be added to the $|f| \times |X||Z|$ and then the result will be reshaped to an $|f| \times |X| \times |Z|$ tensor. After that the tensor will be added to the length-$|X|$ feature vector of each nodes (reshaped to $1 \times 1 \times |X| \times 1$ tensor). In that case, for each $i_j \in f$, the i_k^{th} will produce

$$\begin{bmatrix} -\infty \\ -\infty \\ \cdots \\ [\varphi_{fi}(x_{i_k}, z_f) - m_{f\to i}(x_i) + b_{i_j}(x_{i_j})]_{x_{i_k}=x_{i_j}=1, z_f=1}^{x_{i_k}=x_{i_j}=|X|, z_f=|Z|} \\ \cdots \\ -\infty \end{bmatrix}.$$

The $|f|$ $|f| \times |X| \times |Z|$ tensors will be stacked into an $|f| \times |f| \times |X| \times |Z|$ tensor, and it will be multiplied by the length-$|f|$ one-hot edge feature vector. That will produce

$$\begin{bmatrix} -\infty \\ -\infty \\ \cdots \\ [\varphi_{fi}(x_{i_j}, z_f) - m_{f\to i_j}(x_{i_j}) + b_{i_j}(x_{i_j})]_{x_{i_j}=1, z_f=1}^{x_{i_j}=|X|, z_f=|Z|} \\ \cdots \\ -\infty \end{bmatrix}.$$

Then the max operation over all $i \in f$ will produce edge feature matrix

$$\begin{bmatrix} [\varphi_{fi_1}(x_{i_1}, z_f) - m_{f\to i_1}(x_{i_1}) + b_{i_1}(x_{i_1})]_{x_{i_1}=1,z_f=1}^{x_{i_1}=|X|,z_f=|Z|} \\ [\varphi_{fi_2}(x_{i_2}, z_f) - m_{f\to i_2}(x_{i_2}) + b_{i_2}(x_{i_2})]_{x_{i_2}=1,z_f=1}^{x_{i_2}=|X|,z_f=|Z|} \\ \dots \\ [\varphi_{fi_{|f|}}(x_{i_2}, z_f) - m_{f\to i_n}(x_{i_n}) + b_{i_{|f|}}(x_{i_{|f|}})]_{x_{i_{|f|}}=1,z_f=1}^{x_{i_{|f|}}=|X|,z_f=|Z|} \end{bmatrix}.$$

Then by Proposition 3, we can recover the maximization operation in (42) using an $\mathcal{O}(\log_2 |X|)$-layer neural network with at most $\mathcal{O}(|X|^2 \log_2 |X|)$ hidden units. After that, all the other operations are simple linear operations, and they can be easily encoded in a neural-network without adding any parameter. Thus we can construct an FGNN layer, which produces factor features for each factor f as follows

$$\begin{bmatrix} [b_{f\to i_1}(z_f)]_{z_f=1}^{z_f=|Z|} \\ [b_{f\to i_2}(z_f)]_{z_f=1}^{z_f=|Z|} \\ \dots \\ [b_{f\to i_{|f|}}(z_f)]_{z_f=1}^{z_f=|Z|} \end{bmatrix}.$$

Finally we constructed an FGNN to parameterize the operation in (28a), and this construction also proves Proposition 5 as follows.

Proposition 5. *The operation in* (28a) *can be parameterized by one MPNN layer with* $\mathcal{O}(|X| \max_{c\in\mathcal{C}} |\mathcal{Z}_c|$ *hidden units followed by a* $\mathcal{O}(\log_2 |X|)$*-layer neural network with at most* $\mathcal{O}(|X|^2 \log_2 |X|)$ *hidden units.*

FGNN Layer to recover (28c) Here we construct an FGNN layer to parameterize (28b) and (28c) in order to prove Proposition 6. Using the notation in this section the operation in (28c) can be reformulated as

$$\begin{aligned} m_{f\to i}(x_i) &\leftarrow \max_z [\varphi_{if}(x_i, z_f) + b_{c\to i}(z_f)] \\ b_i(x_i) &\leftarrow \theta_i(x_i) + \sum_{f:i\in f} \max_z [\varphi_{if}(x_i, z_f) + b_{f\to i}(z_f)]. \end{aligned}$$

In previous paragraph, the new factor feature

$$\begin{bmatrix} [b_{f\to i_1}(z_f)]_{z_f=1}^{z_f=|Z|} \\ [b_{f\to i_2}(z_f)]_{z_f=1}^{z_f=|Z|} \\ \dots \\ [b_{f\to i_{|f|}}(z_f)]_{z_f=1}^{z_f=|Z|} \end{bmatrix}.$$

Considering the old factor feature

$$\begin{bmatrix} [\varphi_{fi}(x_{i_1}, z_f)]_{x_{i_1}=1,z_f=1}^{x_{i_1}=|X|,z_f=|Z|} \\ [\varphi_{fi}(x_{i_2}, z_f)]_{x_{i_2}=1,z_f=1}^{x_{i_2}=|X|,z_f=|Z|} \\ \dots \\ [\varphi_{fi}(x_{i_{|f|}}, z_f)]_{x_{i_{|f|}}=1,z_f=1}^{x_{i_{|f|}}=|X|,z_f=|Z|} \end{bmatrix},$$

we can use *broadcasted* addition between these two features to get

$$\begin{bmatrix} [b_{f\to i_1}(z_f)+\varphi_{fi}(x_{i_1},z_f)]_{x_{i_1}=1,z_f=1}^{x_{i_1}=|X|,z_f=|Z|} \\ [b_{f\to i_2}(z_f)+\varphi_{fi}(x_{i_2},z_f)]_{x_{i_2}=1,z_f=1}^{x_{i_2}=|X|,z_f=|Z|} \\ \cdots \\ [b_{f\to i_{|f|}}(z_f)+\varphi_{fi}(x_{i_{|f|}},z_f)]_{x_{i_{|f|}}=1,z_f=1}^{x_{i_{|f|}}=|X|,z_f=|Z|} \end{bmatrix}.$$

After that we have an $|f|\times|X|\times|Z|$ feature tensor for each factor $f\in\mathcal{F}$. By 3, a $\mathcal{O}(\log_2|\mathcal{Z}|)$-layer neural network with at most $\mathcal{O}(|\mathcal{Z}|^2\log_2|\mathcal{Z}|)$ parameters can be used to convert the above feature to

$$\begin{bmatrix} [m_{f\to i_1}(x_{i_1})]_{x_{i_1}=1}^{x_{i_1}=|X|} \\ [m_{f\to i_2}(x_{i_2})]_{x_{i_2}=1}^{x_{i_2}=|X|} \\ \cdots \\ [m_{f\to i_{|f|}}(x_{i_{|f|}})]_{x_{i_{|f|}}=1}^{x_{i_{|f|}}=|X|} \end{bmatrix} \leftarrow \begin{bmatrix} [\max_{z_f}[b_{f\to i_1}(z_f)+\varphi_{fi}(x_{i_1},z_f)]]_{x_{i_1}=1}^{x_{i_1}=|X|} \\ [\max_{z_f}[b_{f\to i_2}(z_f)+\varphi_{fi}(x_{i_2},z_f)]]_{x_{i_2}=1}^{x_{i_2}=|X|} \\ \cdots \\ [\max_{z_f}[b_{f\to i_{|f|}}(z_f)+\varphi_{fi}(x_{i_{|f|}},z_f)]]_{x_{i_{|f|}}=1}^{x_{i_{|f|}}=|X|} \end{bmatrix}.$$

We will use this as the first part of our $\mathcal{M}$ network. For the second part, as we need to parameterize the $\sum_{f:i\in f}\max_z[\varphi_{if}(x_i,z_f)+b_{c\to i}(z_f)]$ from feature $\max_z[\varphi_{if}(x_i,z_f)+b_{c\to i}(z_f)$, by Proposition 4, it will require another linear layer with $\mathcal{O}(\max_{i\in\mathcal{V}}\deg(\mathrm{i})^2|X|^2)$, where $\deg(i)=|\{f|f\in\mathcal{F},i\in f\}|$. After that, the $\mathcal{Q}$ network can be a simple identity mapping, and the FGNN would produce updated messages $m_{f\to i}(x_i)=\max_z[\varphi_{if}(x_i,z_f)+b_{c\to i}(z_f)]$ for each node. Adding these feature with the initial node feature would results new node feature $b_i(x_i)$. Thus by constructing a FGNN layer to parameterize (28b) and (28c) we complete the proof of Proposition 6.

A.5 Example of Recovering Max Product Belief Propagation

We provide a simple example that uses the proposed FGNN to recover Max Product Belief Propagation. Since the Max Product Belief Propagation can be viewed as a continuous mapping between the input log-potentials and the output "beliefs", and our FGNN is acutally a universal approximator for such mapping. In this part, we provide one parametrization of FGNN that can exactly recover Max Product Belief Propagation, but it may not be the only one or the optimal one. Our goal is to design a network that is capabale of recovering traditional inference procedures such as belief propagation, but by learning from data our approach may learn a better inference approach.

Let's consider a simple MAP inference problem over a simple graphical model as follows,

$$\max_{\mathbf{x}}\left[\theta_{1,2}(x_1,x_2)+\theta_{2,3}(x_2,x_3)\right], \tag{44}$$

where each variable $x_i\in\{0,1,\dots N-1\}$, and the log-potentials are all real-valued functions. Then we show the most complicated procedure of (28), that is (28a), can be recovered by a Variable-to-Factor module. In such a Variable-to-Factor (shown in Figure 13) module, in the first layer of $\mathcal{M}(\cdot|\Theta_{FV})$, the edge potentials are mapped into the decomposed log-potentials defined in Lemma 1. This operation only requires a linear transformation. Then the

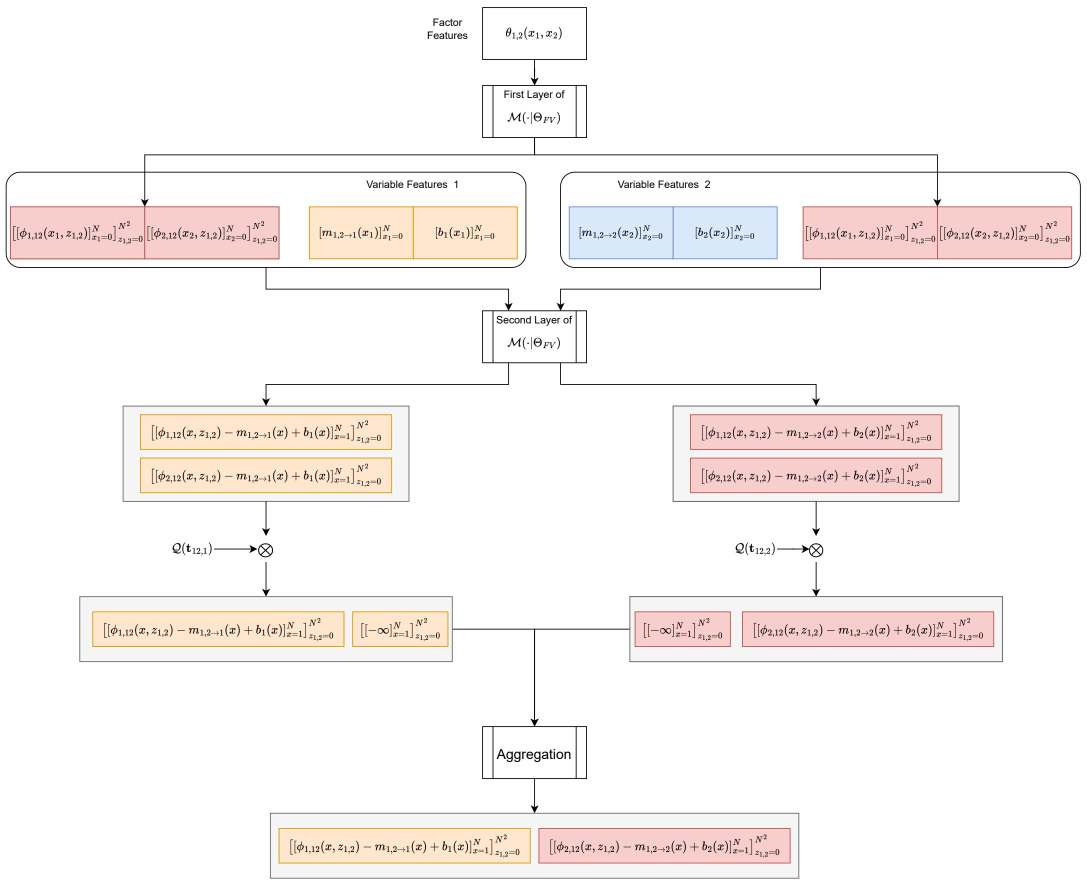

Factor Features
First Layer of $\mathcal{M}(\cdot|\Theta_{FV})$
Variable Features 1
Variable Features 2
Second Layer of $\mathcal{M}(\cdot|\Theta_{FV})$
Aggregation

Figure 13: Example of Variable-to-Factor module that recovers the operations in (28a).

decomposed log-potentials will be concatenated as the input of the second layer of $\mathcal{M}(\cdot|\Theta_{FV})$ (recall that the Variable-to-Factor requires both factor and variable feature as input), then by another linear transformation we can get the term inside the max-operation in (28a), plus a redundant term with the same shape. Then the $\mathcal{Q}$ network, works as a selector, will set the redundant term to $-\infty$, then by the aggregation part, the redundant term will be filtered. Finally, by applying MLP to max over x and do the summation, the factor $\{1, 2\}$ can have a feature vector that consists of $b_{1,2\to1}(z_c)$ and $b_{1,2\to2}(z_c)$.

Appendix B. Experiments

B.1 Additional Information on MAP Inference over PGM

Data We construct four datasets. All variables are binary. The instances start with a chain structure with unary potential on every node and pairwise potentials between consecutive nodes. A higher-order potential is then imposed to every node for the first three datasets.

The node potentials are all randomly generated from the uniform distribution over $[0, 1]$. We use two kinds of pairwise potentials, one randomly generated (as in Table 10), the other encouraging two adjacent nodes to both take state 1 (as in Table 9 and Table 11), i.e. the potential function gives high value to configuration $(1, 1)$ and low value to all other configurations. For example, in Dataset1, the potential value for x_1 to take the state 0 and x_2 to take the state 1 is 0.2; in Dataset3, the potential value for x_1 and x_2 to take the state 1 at the same time is sampled from a uniform distribution over [0, 2].

pairwise potential	$x_2 = 0$	$x_2 = 1$
$x_1 = 0$	0	0.1
$x_1 = 1$	0.2	1

Table 9: Pairwise Potential for Dataset1

pairwise potential	$x_2 = 0$	$x_2 = 1$
$x_1 = 0$	U[0,1]	U[0,1]
$x_1 = 1$	U[0,1]	U[0,1]

Table 10: Pairwise Potential for Dataset2,4

pairwise potential	$x_2 = 0$	$x_2 = 1$
$x_1 = 0$	0	0
$x_1 = 1$	0	U[0,2]

Table 11: Pairwise Potential for Dataset3

For Dataset1,2,3, we additionally add the budget higher-order potential (Martins et al., 2015) at every node; these potentials allow at most k of the 8 variables that are within their scope to take the state 1. For the first two datasets, the value k is set to 5; for the third dataset, it is set to a random integer in {1,2,3,4,5,6,7,8}. For Dataset4, there is no higher-order potential.

As a result of the constructions, different datasets have different inputs for the FGNN; for each dataset, the inputs for each instance are the parameters of the PGM that are not fixed. For Dataset1, only the node potentials are not fixed, hence each input instance is a factor graph with the randomly generated node potential added as the input node feature for each variable node. Dataset2 and Dataset4 are similar in terms of the input format, both including randomly generate node potentials as variable node features and randomly generated pairwise potential parameters as the corresponding pairwise factor node features. Finally, for Dataset3, the variable nodes, the pairwise factor nodes and the high order factor nodes all have corresponding input features.

Architecture We use a multi-layer factor graph neural network with architecture FGNN(64) - Res[FC(64) - FGNN(64) - FC(64)] - MLP(128) - Res[FC(64) - FGNN(64) - FC(128)] - FC(256) - Res[FC(256) - FGNN(64) - FC(256)] - FC(128) - Res[FC(128) - FGNN(64) - FC(128)] - FC(64) - Res[FC(64) - FGNN(64) - FC(64)] - FGNN(2). Here one FGNN(C_{out}) is a FGNN layer with C_{out} as output feature dimension with ReLU (Nair and Hinton, 2010) as activation. One FC(C_{out}) is a fully connected layer with C_{out} as output feature dimension and ReLU as activation. Res[$\cdot$] is a neural network with residual link from its input to output; these additional architecture components can assist learning.

Running Time We report the inference time of one instance and the training time of one epoch for the synthetic datasets in Table 12. The results show that our method runs in a reasonable amount of time.

(μs)	PointNet	DGCNN	AD3 (exact/approx)	Max-Product	MPLP	MPNN	Ours
D1	45 (43)	285 (107)	5 / 5	6	57	131 (72)	144 (75)
D2	–	–	532 / 325	1228	55	131 (72)	341 (162)
D3	–	–	91092 / 1059	4041	55	121 (74)	382 (170)

Table 12: Inference time in microseconds of one instance on synthetic datasets and GPU training time of one epoch in milliseconds (in bracket) for applicable methods.

B.2 Implementation details on MAP Solvers

In the experiment, the AD3 code is from the official code repo [6], which comes with a python interface. For Max-Product algorithm, we use the implementation from libdai and convert the budget higher potential as a table function. For the MPLP algorithm, we implemented it in C++ to directly support the budget higher-order potential. The re-implemented version is compared with the original version [7], and its performance is better than the original one in our experiment. So we provide the result of the re-implemented version.

B.3 Dataset Generation and Training Details of LDPC decoding

Data Each instance of training/evaluation data is generated as follows:

Algorithm 2 Data Generation for LDPC decoding

```
Output: y: a 96-bit noisy signal; SNR_{dB}: signal-to-noise ratio, a scalar
  Uniformly sample a 48-bit binary signal $\mathbf{x}$, where for each $0 < i \leqslant 48$, $P(x_i = 1) = P(x_i = 0) = 0.5$
  Encode $\mathbf{x}$ using the “96.3.963” scheme (MacKay, 2009) to get a 96-bit signal $\mathbf{y}$
  sample $\mathrm{SNR}_{dB} \in \{0, 1, , 2, 3, 4\}$ and $\sigma_b \in \{0, 1, , 23, 4, 5\}$ uniformly
  For each $0 < i \leqslant 96$,uniformly, sample
    • $\eta_i \in \mathcal{U}(0, 1)$,
    • $n_i \in \mathcal{N}(0, \sigma^2)$ s.t. $\mathrm{SNR}_{dB} = 20 \log_{10} 1/\sigma$
    • $z_i \in \mathcal{N}(0, \sigma_b^2)$
  Set noisy signal $\tilde{\mathbf{y}}$ to
    • $\tilde{y}_i = y_i + n_i + \mathbb{I}(\eta_i \leqslant 0.05) z_i$
```

During the training of FGNN, the node feature include the noisy signal $\tilde{\mathbf{y}}$ and the signal-to-noise ratio SNR_{dB}. For FGNN, for each factor f, the vector $[\tilde{y}_i]_{i \in f}$ is provided as feature vector. Meanwhile, for each edge from factor node f to one of its variable node i, the factor feature and the variable node feature are put together to get the edge feature.

Architecture In our FGNN, every layer share the same $\mathcal{Q}$ network, which is 2-layer network as follows MLP(64)-MLP(4). Here the first layer comes with a ReLU activation function and the second layer is with no activation function.

The overall structure of our FGNN is as follows Input - Res[FC(64) - FGNN(64) - FC(64)] - Res[FC(64) - FGNN(64) - FC(64)] - FC(64) - FGNN(64) - FC(128) - FC(256) - FGNN(128) - FC(256) - - Res[FC(256) - FGNN(128) - FC(256)] - FC(128) - FGNN(128) - FC(128) - FC(64) - FGNN(64) - FC(64) - Res[FC(64) - FGNN(64) - FC(64)] - FC(128) - FC(128) - FC(1). In the network, a batch-normalization layer and

6. `https://github.com/andre-martins/AD3`
7. `https://people.csail.mit.edu/dsontag/code/mplp_ver2.tgz`

a ReLU activation function is after each FC layer and FGNN layer except for the last FC layer.

B.4 Additional experiments on Molecular data

B.4.1 Improvement over MPNN when distance information is excluded

In the main paper, we reported the improved performance of FGNN over MPNN on Alchemy dataset. The FGNN is built on MPNN as backend feature extractor. Therefore, its improved performance is the result of capturing dependencies not modeled by MPNN. This begs the question: can higher-order message passing capture more information when the backend MPNN module is further constrained. For this, we limit the input to MPNN module and see if FGNN can further improve its gain with respect to MPNN.

One of the main reasons for the superior performance of MPNN on molecular datasets is that it can capture the 3D geometric structure of the molecule (Chen et al., 2019; Gilmer et al., 2017). MPNN is provided with edge features which include bond type and spatial distance between the pair of atoms. Further, it operates on a complete graph where extra *virtual edges* are added between every pair of atoms with no bond. The edge feature for such *virtual edges* contains the spatial distance between the pair of atoms. Consequently, MPNN can capture 3D geometric structure of the molecule with such a complete graph.

In the following experiments, we evaluate whether higher-order message passing can help capture structure of the molecule in the absence of pairwise distance edge features. We do include 3D atom positions in the node features and hence the information about the geometric structure of the molecule is indirectly provided. Based on the pairwise distance feature, we divide the experimental setup in three categories .

- **Sparse graph without distance:** Input graph is a sparse graph i.e., edge exists only if bond exists between atoms, with edge features containing the bond type without the distance between the pair of atoms.

- **Sparse graph with distance:** Input graph is a sparse graph with edge features containing bond type and distance between the pair of atoms.

- **Complete graph with distance:** Input graph is a complete graph with extra *virtual edges* containing distance information between pair of atoms in the edge. This setup is the standard MPNN model.

In all three cases, regardless of input to the MPNN module, the higher-order message passing works only on the sparse graph.

Results in Table 13 shows that the margin of improvement of FGNN over MPNN is significantly higher when pairwise distance feature is not included in the graph. This suggests MPNN is not able to sufficiently capture the 3D molecular shape in both the cases where sparse graph is used, In such scenarios, capturing higher-order structures with FGNN is more helpful in reducing the MAE. Furthermore, results of both the cases are similar when sparse graph is used and inclusion of pairwise distance in the edge feature does not lead to significant performance gains. Following this, it can be inferred that bond types are indicative of distance between the atoms as well.

Table 13: Comparison of FGNN with MPNN on Alchemy dataset with/without distance information

Target	Sparse graph without distance			Sparse graph with distance			Complete graph with distance		
	MPNN	FGNN	Gain(%)	MPNN	FGNN	Gain(%)	MPNN	FGNN	Gain(%)
μ	0.3546	0.3071	13.39	0.3655	0.3012	17.60	0.1026	0.1041	-1.41
α	0.1971	0.0873	55.68	0.1639	0.0888	45.80	0.0558	0.0451	19.06
ϵ_{homo}	0.1723	0.1281	25.66	0.1539	0.1220	20.73	0.1151	0.1005	12.75
ϵ_{lumo}	0.1280	0.0905	29.31	0.1120	0.0847	24.37	0.0817	0.0664	18.74
Δ_ϵ	0.1266	0.0902	28.75	0.1084	0.0854	21.18	0.0832	0.0692	16.88
$\langle R^2 \rangle$	0.1439	0.0629	56.31	0.2926	0.0629	78.50	0.0271	0.0099	63.47
$ZPVE$	0.1388	0.0412	70.28	0.1020	0.0399	60.80	0.0259	0.0115	55.42
U_0	0.0806	0.0206	74.39	0.0606	0.0236	61.05	0.0131	0.0045	65.80
U	0.0806	0.0206	74.37	0.0606	0.0236	61.05	0.0131	0.0045	65.90
H	0.0806	0.0209	74.07	0.0606	0.0236	60.98	0.0131	0.0045	65.77
G	0.0806	0.0208	74.13	0.0606	0.0236	61.02	0.0131	0.0045	65.77
C_v	0.2177	0.0913	58.04	0.1729	0.0931	46.13	0.0559	0.0481	13.93
MAE	0.1501	0.0818	**45.50**	0.1428	0.0810	**43.24**	0.0499	0.0394	**21.18**

B.4.2 Results of ablation models on QM9 dataset:

In the main paper, we considered ablation models of FGNN based on conditioning of factor parameters. These models are CAT (central atom type), BT (bond type), CABT (central atom and bond type) and CABTA (central atom, bond type and neighbouring atom type). We reported results on Alchemy dataset where we found that conditioning on bond type was sufficient for good performance. To further verify the results, we evaluate the ablation models on QM9 dataset.

Results in Table 14 show that unlike Alchemy dataset, CABT model performs better in almost all the targets on QM9 dataset. This perhaps suggests that in QM9 dataset, higher-order constraints are more centered around the atom and are better captured by having separate parameters for different central atom types. Collectively, the ablation study on QM9 and Alchemy datasets suggests that conditioning the parameters on neighbouring atom type (CABTA) is not helpful and only increases the paramter size. It is sufficient if edge type and central atom type information is directly captured in the model.

References

Sameer Agarwal, Kristin Branson, and Serge Belongie. Higher order learning with graphs. In *Proceedings of the 23rd international conference on Machine learning*, pages 17–24, 2006.

Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. In *Advances in neural information processing systems*, pages 4502–4510, 2016.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan

Table 14: Ablation models regression results on QM9 dataset

Target	FGNN Ablation Models			
	CAT	BT	CABT	CABTA
μ	**0.088**	0.095	0.092	0.097
α	0.192	0.204	**0.183**	0.185
ϵ_{homo}	0.0021	0.00217	**0.00199**	0.00203
ϵ_{lumo}	0.00212	0.0022	**0.00205**	0.00208
Δ_ϵ	0.0029	0.00301	**0.00279**	0.00286
$\langle R^2 \rangle$	2.88	3.01	**2.81**	3.17
$ZPVE$	0.00021	0.00021	**0.00018**	0.00022
U_0	0.1283	0.0995	**0.0907**	0.1649
U	0.1283	0.1029	**0.0907**	0.1650
H	0.1283	0.1008	**0.0906**	0.1649
G	0.1283	0.1011	**0.0906**	0.1651
C_v	0.0868	0.0900	**0.0847**	0.0872
MAE	0.3141	0.3183	**0.2947**	0.3512

Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Mohsen Bayati, Devavrat Shah, and Mayank Sharma. Max-product for maximum weight matching: Convergence, correctness, and lp duality. *IEEE Transactions on Information Theory*, 54(3):1241–1251, 2008.

Lubomir Bourdev and Jitendra Malik. Poselets: Body part detectors trained using 3d human pose annotations. In *2009 IEEE 12th International Conference on Computer Vision*, pages 1365–1372. IEEE, 2009.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

Guangyong Chen, Pengfei Chen, Chang-Yu Hsieh, Chee-Kong Lee, Benben Liao, Renjie Liao, Weiwen Liu, Jiezhong Qiu, Qiming Sun, Jie Tang, et al. Alchemy: A quantum chemistry dataset for benchmarking ai models. *arXiv preprint arXiv:1906.09427*, 2019.

Liang-Chieh Chen, Alexander Schwing, Alan Yuille, and Raquel Urtasun. Learning deep structured models. In *International Conference on Machine Learning*, pages 1785–1794, 2015.

Zhengdao Chen, Lisha Li, and Joan Bruna. Supervised community detection with line graph neural networks. In *International Conference on Learning Representations*, 2018.

Minsu Cho, Jungmin Lee, and Kyoung Mu Lee. Reweighted random walks for graph matching. In *European conference on Computer vision*, pages 492–505. Springer, 2010.

Kamal Choudhary and Brian DeCost. Atomistic line graph neural network for improved materials property predictions. *npj Computational Materials*, 7(1):185, 2021.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Hanjun Dai, Bo Dai, and Le Song. Discriminative embeddings of latent variable models for structured data. In *International conference on machine learning*, pages 2702–2711. PMLR, 2016.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pages 3844–3852, 2016.

David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pages 2224–2232, 2015.

Amir Egozi, Yosi Keller, and Hugo Guterman. A probabilistic approach to spectral graph matching. *IEEE transactions on pattern analysis and machine intelligence*, 35(1):18–27, 2012.

Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

Xiaomin Fang, Lihang Liu, Jieqiong Lei, Donglong He, Shanzhuo Zhang, Jingbo Zhou, Fan Wang, Hua Wu, and Haifeng Wang. Geometry-enhanced molecular representation learning for property prediction. *Nature Machine Intelligence*, 4(2):127–134, 2022.

Yifan Feng, Haoxuan You, Zizhao Zhang, Rongrong Ji, and Yue Gao. Hypergraph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3558–3565, 2019.

M. Fey, J. E. Lenssen, C. Morris, J. Masci, and N. M. Kriege. Deep graph matching consensus. In *International Conference on Learning Representations (ICLR)*, 2020.

Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1263–1272. JMLR. org, 2017.

Amir Globerson and Tommi S Jaakkola. Fixing max-product: Convergent message passing algorithms for map lp-relaxations. In *Advances in neural information processing systems*, pages 553–560, 2008.

Jianwen Jiang, Yuxuan Wei, Yifan Feng, Jingxuan Cao, and Yue Gao. Dynamic hypergraph neural networks. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2635–2641, 2019.

Peter Karkus, David Hsu, and Wee Sun Lee. Qmdp-net: Deep learning for planning under partial observability. In *Advances in Neural Information Processing Systems*, pages 4694–4704, 2017.

Robert H Kassel. *A comparison of approaches to on-line handwritten character recognition*. PhD thesis, Massachusetts Institute of Technology, 1995.

Steven Kearnes, Kevin McCloskey, Marc Berndl, Vijay Pande, and Patrick Riley. Molecular graph convolutions: moving beyond fingerprints. *Journal of computer-aided molecular design*, 30(8):595–608, 2016.

Hyeji Kim, Yihan Jiang, Ranvir Rana, Sreeram Kannan, Sewoong Oh, and Pramod Viswanath. Communication algorithms via deep learning. In *6th International Conference on Learning Representations, ICLR 2018*, 2018.

JinHyung Kim and Judea Pearl. A computational model for causal and diagnostic reasoning in inference systems. In *International Joint Conference on Artificial Intelligence*, pages 0–0, 1983.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

Johannes Klicpera, Janek Groß, and Stephan Günnemann. Directional message passing for molecular graphs. *arXiv preprint arXiv:2003.03123*, 2020.

Pushmeet Kohli and M Pawan Kumar. Energy minimization for linear envelope mrfs. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1863–1870. IEEE, 2010.

Tamara G Kolda and Brett W Bader. Tensor decompositions and applications. *SIAM review*, 51(3):455–500, 2009.

Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

Xiangyang Lan, Stefan Roth, Daniel Huttenlocher, and Michael J Black. Efficient belief propagation with learned higher-order markov random fields. In *European conference on computer vision*, pages 269–282. Springer, 2006.

Marius Leordeanu, Martial Hebert, and Rahul Sukthankar. An integer projected fixed point method for graph matching and map inference. In *Advances in neural information processing systems*, pages 1114–1122, 2009.

Chen Li, Zhen Zhang, Wee Sun Lee, and Gim Hee Lee. Convolutional sequence to sequence model for human dynamics. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5226–5234, 2018.

Maosen Li, Siheng Chen, Yangheng Zhao, Ya Zhang, Yanfeng Wang, and Qi Tian. Dynamic multiscale graph neural networks for 3d skeleton based human motion prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 214–223, 2020.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

Xiaowei Liao, Yong Xu, and Haibin Ling. Hypergraph neural networks for hypergraph matching. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1266–1275, 2021.

Guosheng Lin, Chunhua Shen, Ian Reid, and Anton van den Hengel. Deeply learning the messages in message passing inference. In *Advances in Neural Information Processing Systems*, pages 361–369, 2015.

Guosheng Lin, Chunhua Shen, Anton Van Den Hengel, and Ian Reid. Efficient piecewise training of deep structured models for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3194–3203, 2016.

Yi Liu, Limei Wang, Meng Liu, Yuchao Lin, Xuan Zhang, Bora Oztekin, and Shuiwang Ji. Spherical message passing for 3d molecular graphs. In *International Conference on Learning Representations (ICLR)*, 2022.

Zhi-Yong Liu and Hong Qiao. Gnccp—graduated nonconvexityand concavity procedure. *IEEE transactions on pattern analysis and machine intelligence*, 36(6):1258–1267, 2013.

Zhiyong Liu, Hong Qiao, Xu Yang, and Steven C. H. Hoi. Graph matching by simplified convex-concave relaxation procedure. *International Journal of Computer Vision*, 109(3): 169–186, 2014. URL https://doi.org/10.1007/s11263-014-0707-7.

David MacKay. David mackay's gallager code resources. *Dostupnỳ z URL: http://www.inference.phy.cam.ac.uk/mackay/CodesFiles.html*, 2009.

Wei Mao, Miaomiao Liu, Mathieu Salzmann, and Hongdong Li. Learning trajectory dependencies for human motion prediction. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 9489–9497, 2019.

Wei Mao, Miaomiao Liu, and Mathieu Salzmann. History repeats itself: Human motion prediction via motion attention. In *European Conference on Computer Vision*, pages 474–489. Springer, 2020.

Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. In *Advances in Neural Information Processing Systems*, pages 2153–2164, 2019.

Julieta Martinez, Michael J Black, and Javier Romero. On human motion prediction using recurrent neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2891–2900, 2017.

André FT Martins, Mário AT Figueiredo, Pedro MQ Aguiar, Noah A Smith, and Eric P Xing. AD3: Alternating directions dual decomposition for map inference in graphical models. *The Journal of Machine Learning Research*, 16(1):495–545, 2015.

Joris M Mooij. libdai: A free and open source c++ library for discrete approximate inference in graphical models. *Journal of Machine Learning Research*, 11(Aug):2169–2173, 2010.

Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4602–4609, 2019.

Kevin Murphy, Yair Weiss, and Michael I Jordan. Loopy belief propagation for approximate inference: An empirical study. *arXiv preprint arXiv:1301.6725*, 2013.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Elsevier, 2014.

Brian Potetz and Tai Sing Lee. Efficient belief propagation for higher-order cliques using linear constraint nodes. *Computer Vision and Image Understanding*, 112(1):39–54, 2008.

Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 652–660, 2017.

Stephan Rabanser, Oleksandr Shchur, and Stephan Günnemann. Introduction to tensor decompositions and their applications in machine learning. *arXiv preprint arXiv:1711.10781*, 2017.

Raghunathan Ramakrishnan, Pavlo O Dral, Matthias Rupp, and O Anatole Von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific data*, 1: 140022, 2014.

Michal Rolínek, Paul Swoboda, Dominik Zietlow, Anselm Paulus, Vít Musil, and Georg Martius. Deep graph matching via blackbox differentiation of combinatorial solvers. In *European Conference on Computer Vision*, pages 407–424. Springer, 2020.

Lars Ruddigkeit, Ruud Van Deursen, Lorenz C Blum, and Jean-Louis Reymond. Enumeration of 166 billion organic small molecules in the chemical universe database gdb-17. *Journal of chemical information and modeling*, 52(11):2864–2875, 2012.

Victor Garcia Satorras and Max Welling. Neural enhanced belief propagation on factor graphs. *arXiv preprint arXiv:2003.01998*, 2020.

Vıctor Garcia Satorras, Emiel Hoogeboom, and Max Welling. E (n) equivariant graph neural networks. In *International conference on machine learning*, pages 9323–9332. PMLR, 2021.

Kristof T Schütt, Farhad Arbabzadah, Stefan Chmiela, Klaus R Müller, and Alexandre Tkatchenko. Quantum-chemical insights from deep tensor neural networks. *Nature communications*, 8:13890, 2017.

Zeren Shui and George Karypis. Heterogeneous molecular graph neural networks for predicting molecule properties. In *2020 IEEE International Conference on Data Mining (ICDM)*, pages 492–500. IEEE, 2020.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Paul Swoboda, Carsten Rother, Hassan Abu Alhaija, Dagmar Kainmuller, and Bogdan Savchynskyy. A study of lagrangean decompositions and dual ascent solvers for graph matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1607–1616, 2017.

Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. In *Advances in Neural Information Processing Systems*, pages 2154–2162, 2016.

Veeresh Taranalli. Commpy: Digital communication with python, version 0.5.0, 2020.

Ben Taskar, Carlos Guestrin, and Daphne Koller. Max-margin markov networks. In *Advances in neural information processing systems*, pages 25–32, 2004.

Théo Trouillon, Christopher R Dance, Éric Gaussier, Johannes Welbl, Sebastian Riedel, and Guillaume Bouchard. Knowledge graph completion via complex tensor factorization. *The Journal of Machine Learning Research*, 18(1):4735–4772, 2017.

Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391*, 2015.

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

Runzhong Wang, Junchi Yan, and Xiaokang Yang. Learning combinatorial embedding networks for deep graph matching. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

Runzhong Wang, Junchi Yan, and Xiaokang Yang. Neural graph matching network: Learning lawler's quadratic assignment problem with extension to hypergraph and multiple-graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

Tao Wang, Haibin Ling, Congyan Lang, and Songhe Feng. Graph matching with adaptive and branching path following. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2853–2867, 2018.

Tao Wang, He Liu, Yidong Li, Yi Jin, Xiaohui Hou, and Haibin Ling. Learning combinatorial solver for graph matching. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7568–7577, 2020.

Yair Weiss and William T Freeman. On the optimality of solutions of the max-product belief-propagation algorithm in arbitrary graphs. *IEEE Transactions on Information Theory*, 47(2):736–744, 2001.

Andrew Wrigley, Wee Sun Lee, and Nan Ye. Tensor belief propagation. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3771–3779. JMLR. org, 2017.

Zhenqin Wu, Bharath Ramsundar, Evan N Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S Pappu, Karl Leswing, and Vijay Pande. Moleculenet: a benchmark for molecular machine learning. *Chemical science*, 9(2):513–530, 2018.

Zhirong Wu, Dahua Lin, and Xiaoou Tang. Deep markov random field for image modeling. In *European Conference on Computer Vision*, pages 295–312. Springer, 2016.

Danfei Xu, Yuke Zhu, Christopher B Choy, and Li Fei-Fei. Scene graph generation by iterative message passing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5410–5419, 2017.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

Zhoubo Xu, Puqing Chen, Romain Raveaux, Xin Yang, and Huadong Liu. Deep graph matching meets mixed-integer linear programming: Relax at your own risk? *arXiv preprint arXiv:2108.00394*, 2021.

Naganand Yadati, Madhav Nimishakavi, Prateek Yadav, Vikram Nitin, Anand Louis, and Partha Talukdar. Hypergcn: A new method for training graph convolutional networks on hypergraphs. In *Advances in Neural Information Processing Systems*, pages 1509–1520, 2019.

KiJung Yoon, Renjie Liao, Yuwen Xiong, Lisa Zhang, Ethan Fetaya, Raquel Urtasun, Richard Zemel, and Xaq Pitkow. Inference in probabilistic graphical models by graph neural networks. In *2019 53rd Asilomar Conference on Signals, Systems, and Computers*, pages 868–875. IEEE, 2019.

Tianshu Yu, Runzhong Wang, Junchi Yan, and Baoxin Li. Learning deep graph matching with channel-independent embedding and hungarian attention. In *International conference on learning representations*, 2019.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. In *Advances in neural information processing systems*, pages 3391–3401, 2017.

Andrei Zanfir and Cristian Sminchisescu. Deep learning of graph matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2684–2693, 2018.

Farhad Zarkeshvari and Amir H Banihashemi. On implementation of min-sum algorithm for decoding low-density parity-check (ldpc) codes. In *Global Telecommunications Conference, 2002. GLOBECOM'02. IEEE*, volume 2, pages 1349–1353. IEEE, 2002.

Muhan Zhang and Pan Li. Nested graph neural networks. *Advances in Neural Information Processing Systems*, 34:15734–15747, 2021.

Ruochi Zhang, Yuesong Zou, and Jian Ma. Hyper-sagnn: a self-attention based graph neural network for hypergraphs. *arXiv preprint arXiv:1911.02613*, 2019.

Zhen Zhang and Wee Sun Lee. Deep graphical feature learning for the feature matching problem. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5087–5096, 2019.

Zhen Zhang, Qinfeng Shi, Julian McAuley, Wei Wei, Yanning Zhang, and Anton Van Den Hengel. Pairwise matching through max-weight bipartite belief propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1202–1210, 2016.

Shuai Zheng, Sadeep Jayasumana, Bernardino Romera-Paredes, Vibhav Vineet, Zhizhong Su, Dalong Du, Chang Huang, and Philip HS Torr. Conditional random fields as recurrent neural networks. In *Proceedings of the IEEE international conference on computer vision*, pages 1529–1537, 2015.

Feng Zhou and Fernando De la Torre. Factorized graph matching. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 127–134. IEEE, 2012.