Factor Graph Neural Network

Zhen Zhang[1] **Fan Wu** [2] **Wee Sun Lee**[3]
[1] Australian Institute for Machine Learning & The University of Adelaide, Australia
[2] University of Illinois at Urbana-Champaign
[3] School of Computing, National University of Singapore
zhen.zhang02@adelaide.edu.au fanw6@illinois.edu leews@comp.nus.edu.sg

Abstract

Most of the successful deep neural network architectures are structured, often consisting of elements like convolutional neural networks and gated recurrent neural networks. Recently, graph neural networks (GNNs) have been successfully applied to graph-structured data such as point cloud and molecular data. These networks often only consider pairwise dependencies, as they operate on a graph structure. We generalize the GNN into a factor graph neural network (FGNN) providing a simple way to incorporate dependencies among multiple variables. We show that FGNN is able to represent Max-Product belief propagation, an approximate inference method on probabilistic graphical models, providing a theoretical understanding on the capabilities of FGNN and related GNNs. Experiments on synthetic and real datasets demonstrate the potential of the proposed architecture.

1 Introduction

Deep neural networks are powerful 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 [7, 34, 35] have recently been successfully used with graph-structured data to capture pairwise dependencies between variables and to propagate the information to the entire graph.

The dependencies in the real-world are often beyond pairwise connections. *E.g.*, in the LDPC encoding, the bits of a signal are grouped into several clusters. In each cluster, the sum of all bits should be equal to zero [36]. Then in the decoding procedure, these constraints should be respected. In this paper, we show that the GNN can be naturally extended to capture dependencies over multiple variables by using the factor graph 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. We call a neural network formed from the factor graph a factor graph neural network (FGNN).

Factor graphs have been used extensively to specify Probabilistic Graph Models (PGMs) for modeling dependencies among multiple random variables. In PGMs, the specification or learning of the model is usually separate from the inference process. Approximate inference algorithms such as Belief Propagation which is often used, since inference over PGMs are often NP-hard. Unlike PGMs, graph neural networks usually learn a set of latent variables and the inference procedure at the same time in an end-to-end manner; the graph structure only provides information on the dependencies along which information propagates. For problems where domain knowledge is weak, or where approximate inference algorithms do

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

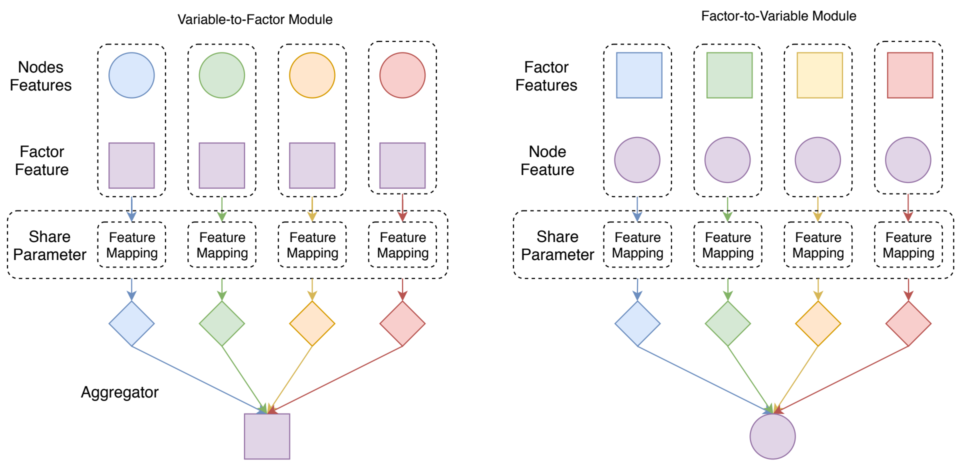


Figure 1: The structure of the Factor Graph Neural Network (FGNN): the Variable-to-Factor (VF) module is shown on the left while the Factor-to-Variable (FV) module is shown on the right.

poorly, being able to learn an inference algorithm jointly with the latent variables, specifically for the target data distribution, often produces superior results.

We take the approach of jointly learning the inference algorithm and latent variables in developing the 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. We show that the FGNN is able to exactly parameterize the Max-Product Belief Propagation, which is a widely used approximate *maximum a posteriori* (MAP) inference algorithm for PGMs. Theoretically, this shows that FGNN is at least as powerful as Max-Product and hence can solve problems solvable by Max-Product, *e.g.*, [2, 11].

The simplest representation of PGMs uses a tabular potential for the factors. Unfortunately, its size grows exponentially with the number of variables in the factors, which makes higher order tabular factors impractical. We design FGNN to naturally allow approximation of the factors by parameterizing factors in terms of the maximum of a set of rank-1 tensors. The 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 may be small for easier problems. Using this representation, the size of the FGNN that can simulate Max-Product grows polynomially with the number of rank-1 tensors in approximating the factors, giving a practical approximation scheme that can be learned from data.

The theoretical relationship with Max-Product provides understanding on the representational capabilities of GNNs in general, and of FGNN in particular. From the practical perspective, the factor graph provides a flexible way for specifying 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 minimums. We performed experiments to explore the practical potential of FGNN. FGNN performed well on a synthetic PGM inference problem with constraints on the number of elements that may be present in subsets of variables. We also experimented with applying FGNN on the LDPC decoding and long term human motion prediction. We outperform the standard LDPC decoding method under some noise conditions and achieve state-of-the-art results on human motion prediction, demonstrating the potential of the architecture.

2 Background

Probabilistic Graph 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 $x_i \in X$, each vertex $c \in \mathcal{C}$ is associated with a function f_c and an edge connects a variable vertex i to factor vertex c if f_c depends on x_i.

Let $\mathbf{x}$ be the set of all variables and let $\mathbf{x}_c$ be the subset of variables that f_c depends on. Denote the set of indices of variables in $\mathbf{x}_c$ by $s(c)$. We consider discrete PGM as follows

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

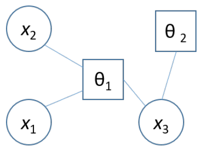


Figure 2: A factor graph where f_1 depends on x_1, x_2, and x_3 while f_2 depends on x_3.

where $\exp(\theta_c(\cdot))$, $\exp(\theta_i(\cdot))$ are positive functions called potentials (with $\theta_c(\cdot)$, $\theta_i(\cdot)$ as the corresponding log-potentials) and Z is a normalizing constant. The goal of *maximum a posteriori* (MAP) inference [16] is to find the assignment which maximizes $p(\mathbf{x})$, that is

$$\mathbf{x}^* = \operatorname*{argmax}_{\mathbf{x}} \sum_{c\in\mathcal{C}}\theta_c(\mathbf{x}_c) + \sum_{i\in\mathcal{V}}\theta_i(x_i). \quad (2)$$

As Eq. (2) is NP-hard in general [29], approximation are often required. One common method is Max-Product Belief Propagation, which is an iterative method formulated as

$$b_i(\mathbf{x}_i) = \theta_i(x_i) + \sum_{c: i\in s(c)} m_{c\to i}(x_i); \quad m_{c\to i}(x_i) = \max_{\hat{\mathbf{x}}_c:\hat{x}_i = x_i}\big[\theta_c(\hat{\mathbf{x}}_c) + \sum_{i'\in s(c), i'\neq i} b_{i'}(\hat{x}_{i'})\big]. \quad (3)$$

Max-product type algorithms are fairly effective in practice, achieving moderate accuracy in various problems [6, 8, 32].

Related Works Various graph neural network models have been proposed for graph structured data, including methods based on the graph Laplacian [3, 4, 13], gated networks [18], and various other neural networks structures for updating the information [1, 5, 9, 28]. Gilmer et al. [7] show that these methods can be viewed as applying message passing on pairwise graphs and are special cases of Message Passing Neural Networks (MPNNs).

In this work, we seek to go beyond pairwise interactions by using message passing on factor graphs. Recent works on the expressive power of graph neural networks have also consider using higher order networks, e.g. Morris et al. [25] and Maron et al. [21] consider networks based on higher order Weisfeiler-Lehman tests that can be used for testing graph isomorphism. In contrast to graph isomorphism, FGNN builds on probabilistic graphical models, which provide a rich language allowing the designer to specify prior knowledge in the form of pairwise as well as higher order dependencies in a factor graph.

3 Factor Graph Neural Network

Previous works on graph neural networks focus on learning pairwise information exchanges. The Message Passing Neural Network (MPNN) [7] provides a framework for deriving different graph neural network algorithms by modifying the message passing operations. We aim at enabling the network to efficiently encode higher order features and to propagate information between higher order factors and the nodes by performing message passing on a factor graph. We describe the FGNN network and show that for specific settings of the network parameters we obtain the Max-Product Belief Propagation algorithm for the corresponding factor graph.

3.1 Factor Graph Neural Network

First we give a brief introduction to the Message Passing Neural Network (MPNN), and then we propose one MPNN architecture which can be easily extended to a factor graph version. Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{N})$, where $\mathcal{V}$ is the node set and $\mathcal{N}$ is the adjacency list, assume that each node is associated with a feature $\mathbf{f}_i$ and each edge $(i, j) : i \in \mathcal{V},\ j \in \mathcal{N}(i)$ is associated with an edge feature $\mathbf{e}_{ij}$. Then a message passing neural network layer is defined in [7] as

$$\mathbf{m}_i = \sum_{j \in \mathcal{N}(i)} \mathcal{M}(\mathbf{f}_i, \mathbf{f}_j, \mathbf{e}_{ij}), \qquad \tilde{\mathbf{f}}_i = \mathcal{U}_t(\mathbf{f}_i, \mathbf{m}_i), \tag{4}$$

where $\mathcal{M}$ and $\mathcal{U}$ are usually parameterized by neural networks. The summation in (4) can be replaced with other aggregator, *e.g.*, maximization [31]. The main reason to use maximization is that summation may be corrupted by a single outlier, while maximization is more robust. Thus in our paper we also use the maximization as aggregator. There are also multiple choices of the architecture of $\mathcal{M}$ and $\mathcal{U}$. We propose an MPNN architecture as follows

$$\tilde{\mathbf{f}}_i = \max_{j \in \mathcal{N}(i)} \mathcal{Q}(\mathbf{e}_{ij})\, \mathcal{M}(\mathbf{f}_i, \mathbf{f}_j), \tag{5}$$

where $\mathcal{M}$ maps feature vectors to a length-n feature vector, and $\mathcal{Q}(\mathbf{e}_{ij})$ maps $\mathbf{e}_{ij}$ to a $m \times n$ matrix. Then by matrix multiplication and aggregation a new length-m feature is generated.

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:     $\tilde{\mathbf{g}}_c = \max_{i:(c,i) \in \mathcal{E}} \mathcal{Q}(\mathbf{t}_{ci} \,|\Phi_{\mathrm{VF}})\, \mathcal{M}([\mathbf{g}_c, f_i] |\Theta_{\mathrm{VF}})$
3: Factor-to-Variable:
4:     $\tilde{f}_i = \max_{c:(c,i) \in \mathcal{E}} \mathcal{Q}(\mathbf{t}_{ci} \,|\Phi_{\mathrm{FV}})\, \mathcal{M}([\ \mathbf{g}_c, f_i] |\Theta_{\mathrm{FV}})$
```

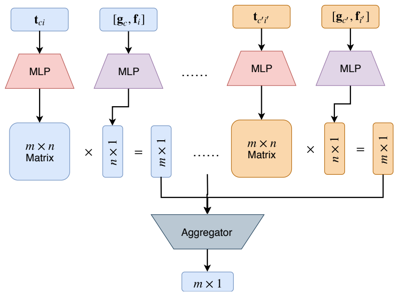


Figure 3: **Left:** The pseudo code for the FGNN layer. Here the Variable-to-Factor (VF) module and the Factor-to-Variable (FV) modules are MPNN layers with similar structure but different parameters. **Right:** The detailed architecture for our VF or FV module.

The MPNN encodes unary and pairwise edge features, but higher order features are not directly encoded. Thus we extend the MPNN by introducing extra factor nodes. 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 $[t_{ci}]$. Then, the Factor Graph Neural Network layer on $\mathcal{G}$ can be extended from (5) as shown in Figure 3 and 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.

We use the same architecture for sending the messages from variables to factors as well as for the messages from factors to variables. If the aim is only to simulate the Max-Product algorithm, it would be more direct to have different architectures for the two types of messages. However, having the same architecture is simpler to implement. In addition, it is also possible to have a variant where messages from variables and factors are sent simultaneously instead of alternately; in this case we simply have a MPNN on a bipartite (factor) graph with the same structure for the variable and factor nodes.

3.2 FGNN for Max-Product Belief Propagation

In this section, we prove that a widely used approximate inference algorithm, Max-Product Belief Propagation can be exactly parameterized by the FGNN. The sketch of the proof is as follows. First we show that any higher order potentials can be 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.

In general, the size of a potential grows exponentially with the number of variables that it depends on. In that case the size of FGNN may explode. However, if the potential can be well approximated as a moderate number of rank-1 tensors, the corresponding FGNN will also be of moderate size. In practice, the potential functions may be unknown and only features of the of the factor nodes are provided; FGNN can learn the approximation from data, potentially exploiting regularities such as low rank approximations if they exist.

Tensor Decomposition For discrete variables $x_1, \ldots, x_n$, a rank-1 tensor is a product of univariate functions of the variables $\prod_{i=1}^{n} \phi_i(x_i)$. A tensor can always be decomposed as a finite sum of rank-1 tensors [15]. This has been used to represent potential functions, e.g. in [33], in conjunction with sum-product type inference algorithms. 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 [14].

Lemma 1 ([14])**.** *Given an arbitrary potential function $\phi_c(\mathbf{x}_c)$, there exists a variable $z_c \in \mathcal{Z}_c$ with $|\mathcal{Z}_c| < \infty$ and a set of univariate potentials $\{\phi_{ic}(x_i, z_c) | i \in c\}$, s.t.*

$$\log \phi_c(\mathbf{x}_c) = \log \max_{z_c} \prod_{i \in s(c)} \phi_{ic}(x_i, z_c) = \max_{z_c} \sum_{i \in s(c)} \varphi_{ic}(x_i, z_c), \tag{6}$$

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

Using ideas from [14], 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.*

The proof of Proposition 2 and the following propositions can be found in the supplementary material. With the decomposed higher order potential, one iteration of the Max-Product (3) can be rewritten using the following two equations:

$$b_{c \to i}(z_c) = \sum_{i' \in s(c), i' \neq i} \max_{x'_i} \left[\varphi_{i'c}(x_{i'}, z_c) + b_{i'}(x_{i'}) \right], \tag{7a}$$

$$b_i(x_i) = \theta_i(x_i) + \sum_{c : i \in s(c)} \max_{z} [\varphi_{ic}(x_i, z_c) + b_{c \to i}(z_c)]. \tag{7b}$$

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, 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 (7a) (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 $\mathbf{X} \in \mathbb{R}^{m \times n}$ with x_{ij} as its entry in the i^{th} row and j^{th} column, the feature mapping operation $\hat{\mathbf{x}} = [\max_j x_{ij}]_{i=1}^{m}$ can be exactly parameterized with a $2\log_2 n$-layer neural network with at most $\mathcal{O}(n^2 \log_2 n)$ parameters.*

Following the maximization operations, Eq. (7a) 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 (7a) 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 (7a). Hence, equation (7a) 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.

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

Eq. (7b) can be implemented in the same way as (7a) 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 (7b). 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 (7). Firstly, by Proposition 3, we can construct the Variable-to-Factor module using the following proposition.

Proposition 5. *The operation in* (7a) *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* (7b) *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 theory result in this paper as follows.

Corollary 7. *The max-product algorithm in* (3) *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 $\max_{c\in\mathcal{C}} |c|$.*

4 Experiments

In this section, we evaluate the models constructed using FGNN for three types of tasks: MAP inference over higher order PGMs, LDPC decoding and human motion prediction.

4.1 MAP Inference over PGMs

Data We construct four synthetic datasets (D1, D2, D3 and D4) for this experiment. All models start with a length-30 chain structure with binary-states nodes with node potentials randomly sampled from $\mathcal{U}[0, 1]$, and the pairwise potentials encourage two adjacent nodes to take state 1, *i.e.*, it gives high value to configuration $(1, 1)$ and low value to others. 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 [23] 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 randomly in D3. In D4, there is no higher order potential at all.

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 MPNN [7] as well as several MAP inference solver, namely AD3 [23] which solves a linear programming relaxation using subgradient methods, Max-Product Belief Propagation [32], implemented by [24], and a convergent version of Max-Product – MPLP [8], also based on a linear programming relaxation. The approximate inference algorithms are run with the

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).

	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)
D4	**100** (6 / 5)	**100** (6)	99.9±0.0005 (0.04)	91.2±0.0005 (137)	98.0±0.0003 (216)

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 [23].

Architecture and training details In this task, we use a factor graph neural network consisting of 8 FGNN layers (the details is provided in the supplementary file). The model is implemented using pytorch [27], trained with Adam optimizer [12] with initial learning rate $\text{lr} = 3 \times 10^{-3}$ and after each epoch, lr is decreased by a factor of 0.98. All the models in Table 1, were trained for 50 epoches after which all models achieve convergence.

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

Traditional method including Max-Product, MPLP perform poorly on D1, D2 and D3. In these even though FGNN can emulate traditional Max-Product, it is better to learn a different inference algorithm. AD3 have better performance than others, but 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.

We also did a small ablation study on the size of the FGNN high order potentials (HOPs) using D1 and D2. 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 size of the HOPs are set correctly.

4.2 LDPC Decoding

The low-density parity check (LDPC) codes is widely used in wired and wireless communication, where the decoding can be done by sum/max-product belief propagation [36].

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"[19]. Then a noisy signal $\tilde{\mathbf{y}}$ is obtained by transferring $\mathbf{y}$ through a channel with 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 [10] 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 trained model.

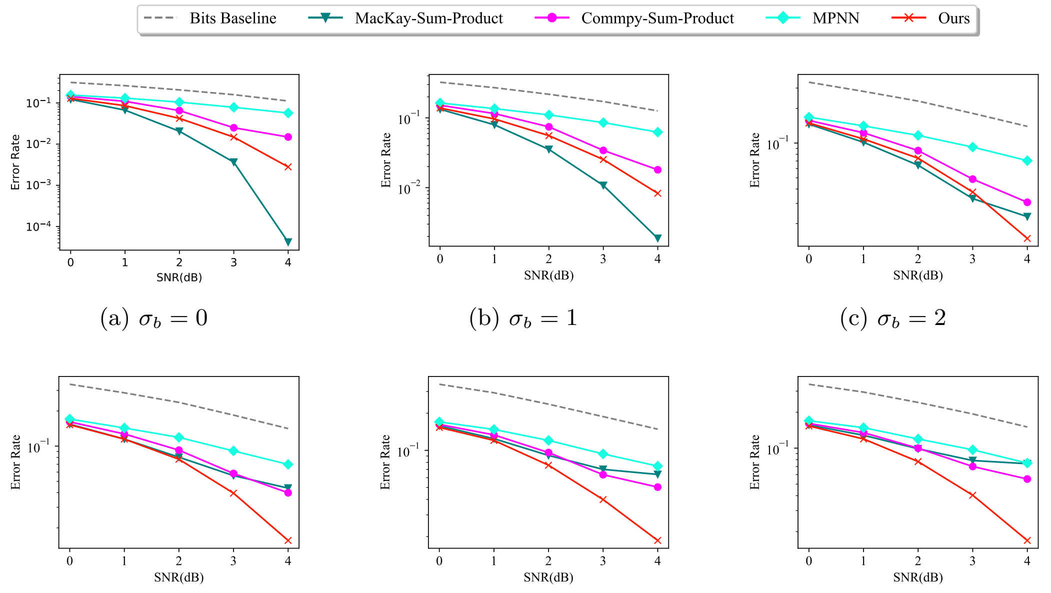
In LDPC decoding, the SNR_{dB} is usually assumed to be known and fixed, and the burst noise is often unexpected and its parameters are unknown to the decoder. Thus for learning based methods and traditional LDPC decoding method, the noisy signal $\tilde{\mathbf{y}}$ and the SNR_{dB} are provided as input. In our experiments, the baselines includes bits decoding, sum-product based LDPC decoding and the Message Passing Neural Networks (MPNN).

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 supplementary file). The model is implemented using pytorch [27], trained with Adam optimizer [12] with initial learning

[1]Additional experiment on tree can be found in Appendix B.3 along with details on all experiments.

Table 2: Long-term prediction error (the smaller the better) of joint angles (top) and 3D joint positions (bottom) on H3.6M

	Walk		Eating		Smoking		Discussion		Average	
milliseconds	560	1000	560	1000	560	1000	560	1000	560	1000
convSeq2Seq[17]	N/A	0.92	N/A	1.24	N/A	1.62	N/A	1.86	N/A	1.41
GNN[20]	0.65	0.67	0.76	1.12	0.87	1.57	1.33	1.70	0.90	1.27
Ours	0.67	0.70	0.76	1.12	0.88	1.57	1.35	1.70	0.91	1.27
convSeq2Seq[17]	69.2	81.5	71.8	91.4	50.3	85.2	101.0	143.0	73.1	100.3
GNN[20]	55.0	60.8	68.1	79.5	42.2	70.6	93.8	119.7	64.8	82.6
Ours	44.1	53.5	59.5	73.0	33.0	61.9	86.9	113.5	55.9	75.5



(a) $\sigma_b = 0$ (b) $\sigma_b = 1$ (c) $\sigma_b = 2$

(d) $\sigma_b = 3$ (e) $\sigma_b = 4$ (f) $\sigma_b = 5$

Figure 4: Experimental results on LDPC decoding.

rate $\text{lr} = 1 \times 10^{-2}$ and after every 10000 samples, lr is decreased by a factor of 0.98. After training over 10^8 samples, the training loss converges. For MPNN, we use a 8 layer MPNN, and the same training protocol is used.

Results We compare FGNN with two public available LDPC decoder MacKay-Sum-Product [19] and Commpy-Sum-Product [30]. Both the two decoder 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. 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 4. The MacKay-Sum-Product [19] is known to be near optimal for Gaussian noise and thus its performance is the best for lower burst noise level. The Commpy-Sum-Product have better performance than MacKay for high burst noise, but worse performance for low burst noise. Our FGNN always perform better than the Commpy-Sum-Product and MPNN, it achieves comparable but lower performance than the MacKay-Sum-Product for low burst noise level(0-2dB), and outperforms all other methods for high burst noise level(3-5dB).

4.3 Human Motion Prediction

The human motion prediction aims at predicting 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 human motion prediction problem for the skeleton data, where the angle and 3d position of each joints 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.

Architecture and training details We train our model on the Human3.6M dataset using the standard training-val-test split as previous works [17, 20, 22], and we train and evaluate our model using the same protocol as [20] (For details, see the supplementary file).

Results The results is provided in Table 2. For angle error, our FGNN model achieves similar results compared to the previous state-of-the-art GNN-based method [20], while for 3D position error, our model achieves superior performance. This is because compared to pairwise GNN, our model captures better higher order structural prior.

5 Conclusion

We extend graph neural networks to factor graph neural networks, enabling the network to capture higher order dependencies among the variables. The factor graph neural networks can represent the execution of the Max-Product algorithm on probabilistic graphical models, providing theoretical understanding on the representation power of graph neural networks. The factor graph 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 structural bias into learning and inference problems.

Broader Impact

Our work on the factor graph neural networks aims to make it easier to effectively specify structural inductive biases in the form of dependencies among sets of variables. This will impact on learning algorithms on structured data, particularly graph structured data. On the positive side, with well specified inductive biases, more effective learning would be possible on applications that require structured data. These include data with physical constraints such as human motion data, as well as data with abstract relationships such as social network data. On the negative side, in applications on some types of data such as social network data, better inference could mean less privacy. Research, guidelines, and possibly regulations on privacy can help to mitigate the negative effects.

Acknowledgements

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.

References

[1] 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.

[2] 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.

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

[4] 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.

[5] 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.

[6] Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient belief propagation for early vision. *International journal of computer vision*, 70(1):41–54, 2006.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

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

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

[14] 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.

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

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

[17] 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.

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

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

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] Solomon Eyal Shimony. Finding maps for belief networks is np-hard. *Artificial Intelligence*, 68(2):399–410, 1994.

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

[31] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic Graph CNN for Learning on Point Clouds. *ACM Transactions on Graphics (TOG)*, 2019.

[32] 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.

[33] Andrew Wrigley, Wee Sun Lee, and Nan Ye. Tensor belief propagation. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3771–3779, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR. URL `http://proceedings.mlr.press/v70/wrigley17a.html`.

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

[35] 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.

[36] 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.

Factor Graph Neural Net—Supplementary File

A Proof of propositions

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

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

$$, \qquad [\mathbf{f}_1, \mathbf{f}_2, \dots, \mathbf{f}_n] = [\max_i \hat{f}_{i0}, \max_i \hat{f}_{i1}, \dots, \max_i \hat{f}_{i,mn}].$$

Proof. Let

$$\mathbf{Q}_i = \left[\underbrace{\mathbf{0}^{m\times m}, \dots, \mathbf{0}^{m\times m}}_{i-1 \text{ matrices}}, \mathbf{I}, \underbrace{\mathbf{0}^{m\times m}, \dots, \mathbf{0}^{m\times m}}_{n-i \text{ matrices}}\right]^\top, \tag{8}$$

then we have that

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

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

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

$$\theta_{ic}(x_i, z_c) = \begin{cases} \frac{1}{|s(c)|} \log \phi_c(\mathbf{x}_c^{z_c}), & \text{if } \hat{x}_i = x_i^{z_c}, \\ -c_{x_i,z_c}, & \text{otherwise,} \end{cases} \tag{9}$$

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

$$\log \phi_c(\mathbf{x}_c) = \max_{z_c} \sum_{i \in s(c)} \theta_{ic}(x_i, z_c) \tag{10}$$

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

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

and the higher order potential can be organized as vector $\mathbf{g}_c = [\log \phi_c(\mathbf{x}_c^0), \log \phi_c(\mathbf{x}_c^1), \dots, \log \phi_c(\mathbf{x}_c^{k^n-1})]$. Then for each i the item $\theta_{ic}(x_i, z_c)$ in (9) have k^{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-k^{n+1} vector $\mathbf{f}_{ic}$, then we define a $k^{n+1} \times k^n$ 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. $\square$

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.

$$b_{c\to i}(z_c) = \sum_{i' \in s(c), i' \neq i} \max_{x_i'} \left[\log \phi_{i'c}(x_{i'}, z_c) + b_{i'}(x_{i'})\right], \tag{11a}$$

$$b_i(x_i) = \theta_i(x_i) + \sum_{c: i \in s(c)} \max_z [\log \phi_{ic}(x_i, z_c) + b_{c\to i}(z_c)]. \tag{11b}$$

In the max-product updating procedure, we should keep all the decomposed $\log \phi_{i'c}(x_{i'}, z_c)$ 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 $\log \phi_{i'c}(x_{i'}, z_c)$ and $\theta_i(x_i)$. The other blocks are used to updating $b_{c\to i}(z_c)$ 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 $\mathbf{X} \in \mathbb{R}^{m\times n}$ with x_{ij} as its entry in the i^{th} row and j^{th} column, the feature mapping operation $\hat{\mathbf{x}} = [\max_j x_{ij}]_{i=1}^m$ can be exactly parameterized with a $2\log_2 n$-layer neural network with Relu as activation function and at most $2n$ hidden units.*

Proof. Without loss of generality we assume that $m = 1$, and then we use x_i to denote x_{1i}. When $n = 2$, 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 $n = 2^k$, the proposition is satisfied. Then for $n = 2^{k+1}$, we can find $\max(x_1, \ldots, x_{2^k})$ and $\max(x_{2^k+1}, \ldots, x_{2^{k+1}})$ using two network with $2k$ layers and at most 2^{k+1} hidden units. Stacking the two neural network together would results in a network with $2k$ layers and at most 2^{k+2}. Then we can add another 2 layer network with 3 hidden units to find $\max(\max(x_1, \ldots, x_{2^k}), \max(x_{2^k+1}, \ldots, x_{2^{k+1}}))$. Thus by mathematical induction the proposition is proved. $\square$

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* $\mathbf{X} \in \mathbb{R}_{\geqslant 0}^{m\times n}$ *with* x_{ij} *as its entry in the* i^{th} *row and* j^{th} *column, there exists a constant tensor* $\mathbf{W} \in \mathbb{R}^{m\times n\times mn}$ *that can be used to transform* $\mathbf{X}$ *into an intermediate representation* $y_{ik} = \sum_{ij} x_{ij} w_{ijk}$*, such that after maximization operations are done to obtain* $\hat{y}_k = \max_i y_{ik}$*, we can use another constant matrix* $\mathbf{Q} \in \mathbb{R}^{n\times mn}$ *to obtain*

$$[\sum_i x_{ij}]_{j=1}^n = \mathbf{Q}[\hat{y}_k]_{k=1}^{mn}. \tag{12}$$

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. $\square$

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 (2), the transformation as follows would make the log-potentials non-negative without changing the final MAP assignment,

$$\tilde{\theta}_i(x_i) = \theta_i(x_i) - \min_{x_i} \theta_i(x_i), \qquad \tilde{\theta}_c(\mathbf{x}_c) = \theta_c(\mathbf{x}_c) - \min_{\mathbf{x}_c} \theta_c(\mathbf{x}_c). \tag{13}$$

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.1 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

$$\varphi_f(\mathbf{x}_f) = \max_{z_f \in \mathcal{Z}} \sum_{i \in f} \varphi_{fi}(x_i, z_f), \tag{14}$$

   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 5.

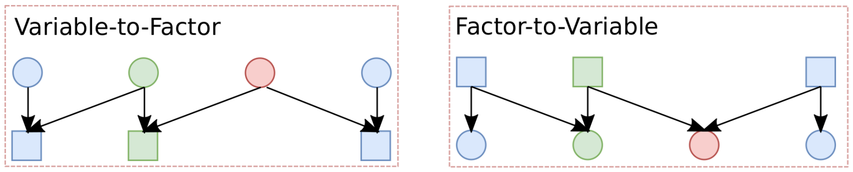


Figure 5: Directed bipartite graph for constructing FGNN layers. In the Variable-to-Factor sub-graph, each factor receives the messages from the same number of nodes. On the other hand, for each Factor-to-Variable sub-graph, each nodes may receives messages from different number of factors.

FGNN Layer to recover (7a) Here we construct an FGNN layer to produce all $b_{f\to i}(z_f)$. First we reformulate (7a) as

$$\begin{aligned} b_{f\to i}(z_f) &= \tilde{\varphi}_f(z_f) - \max_{x_i}[\varphi_{if}(x_i, z_f) + b_i(x_i)], \\ \tilde{\varphi}_c(z_f) &= \sum_{i\in f} \max_{x_i}[\varphi_{if}(x_i, z_f) + b_i(x_i)]. \end{aligned} \tag{15}$$

Here we use the Variable-to-Factor sub-graph to implement (15). 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 $f = [i_1, i_2, \dots, i_{|f|}]$, then for some $i_j \in f$, the associated feature vector is as length-$|f|$ one-hot vector as follows

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

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

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

Then we construct an MPNN

$$\tilde{\mathbf{f}}_i = \max_{i\in f} \mathcal{Q}(\mathbf{e}_{f\to i})\, \mathcal{M}(\mathbf{f}_i, \mathbf{f}_f), \tag{16}$$

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

$$\begin{bmatrix} -\infty \\ -\infty \\ \dots \\ [\varphi_{fi}(x_{i_j}, z_f)]_{x_{i_j}=1, z_f=1}^{x_{i_j}=|X|, z_f=|Z|} \\ \dots \\ -\infty \end{bmatrix}.$$

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

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

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

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

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

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

Then by Proposition 3, we can recover the maximization operation in (15) 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

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

Finally we constructed an FGNN to parameterize the operation in (7a), and this construction also proves Proposition 5 as follows.

Proposition 5. *The operation in* (7a) *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 (7b) Here we construct an FGNN layer to parameterize (7b) in order to prove Proposition 6. Using the notation in this section the operation in (7b) can be reformulated as

$$b_i(x_i) = \theta_i(x_i) + \sum_{f: i\in f} \max_z [\varphi_{if}(x_i, z_f) + b_{c\to i}(z_f)].$$

In previous paragraph, the new factor feature

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

Considering the old factor feature

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

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

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

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 feature $\sum_{f:i\in f}\max_z[\varphi_{if}(x_i,z_f)+b_{c\to i}(z_f)]$ 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 (7b) we complete the proof of Proposition 6.

B Experiments

B.1 Additional Ablation Study

Aggregation Function In the Message Passing Neural Network module, various aggregation function such as "max", "sum" or "average" can be used. In our implementation, we choose the "max" aggregation function because theoretically "max" is invariant to the duplication of a element in the set, while "sum" or "average" is not. In real applications such as human motion prediction, different factor may have different size, but for better parallelization we may need to pad all factor to the same size. In this case, we may simply duplicate a node in factor to do this. We replaced the "max" aggregation with "sum" aggregation in the LDPC experiment and typical result is shown in Figure 6, where both algorithm achieve almost the same performance.

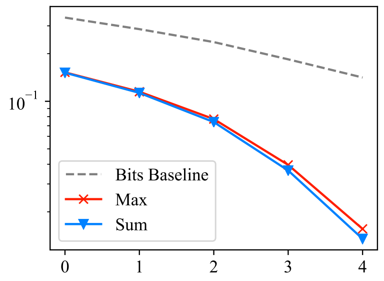


Figure 6: Comparison of "sum" and "max" aggregation.

B.2 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 4), the other encouraging two adjacent nodes to both take state 1 (as in Table 3 and Table 5), 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 3: 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 4: 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 5: Pairwise Potential for Dataset3

For Dataset1,2,3, we additionally add the budget higher order potential [23] 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 [26] 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 6. 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)
D4	–	–	6 / 5	6	0.04	137 (71)	216 (101)

Table 6: 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.3 Experiment on tree structured PGM

Apart from the chain structured PGM in Section 4.1, we also have an additional experiment on tree structured PGM. The training set includes 90000 different PGM as randomly generated binary trees whose depth are between 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 the 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 is generated in the same way as the training set. The experiment result is shown in Table 7.

	AD3	Max-Product	MPLP	MPNN	Ours
Agreement on MAP	1.0	1.0	0.9997	–	0.9835

Table 7: Experimental result on tree structured PGM.

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 4.1, 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.

B.4 Testing on novel graph structures for synthetic data

We conducted a new 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 Dataset3). The result is in Table 8, which shows that the model trained on fixed size MRF can be generalized to MRF with different graph structures.

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

Table 8: Accuracy on dataset with different chain size.

B.5 Implementation details on MAP Solvers

In the experiment, the AD3 code is from the official code repo [2], 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 [3], and its performance is better than the original one in our experiment. So we provide the result of the re-implemented version.

B.6 Dataset Generation and Training Details of LDPC decoding

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

During the training of MPNN and FGNN, the node feature include the noisy signal $\tilde{\mathbf{y}}$ and the signal-to-noise ratio SNR_{dB}. For MPNN, no other feature are provided, while 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.

[2] `https://github.com/andre-martins/AD3`

[3] `https://people.csail.mit.edu/dsontag/code/mplp_ver2.tgz`

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 [19] 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
      • $\tilde{y}_i = y_i + n_i + \mathbb{I}(\eta_i \leqslant 0.05) z_i$
```

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

B.7 Details of Human Motion Prediction

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. [20]'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[17, 20], 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 [20], and our FGNN model also strictly followed the training protocol of [20].

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