Policy Message Passing: Modeling Trajectories for Probabilistic Graph Inference

Zhiwei Deng [1] **Xingguo Li** [1] **Greg Mori** [2]

Abstract

Learning to perform flexible reasoning over multiple variables is of fundamental importance for various tasks in machine learning. Graph neural network is an effective framework for building inference processes among variables. A powerful graph-structured neural network architecture should operate on graphs through two core components: (1) complex message functions designed to model relations between nodes; (2) a flexible information aggregation process that executes the reasoning through passing messages. However, despite the efforts on message designs, existing graph neural networks have limited power of systematically modeling flexible reasoning over graphs. In this paper, we propose the *Policy Message Passing (PMP)* algorithm, which takes a probabilistic perspective and reformulates the whole information aggregation as stochastic sequential processes. PMP is built upon the Variational Inference framework and defines a set of automatic agents that observe the states and history to perform actions over the graphs. Theoretical interpretations are provided to show that our algorithm can achieve improved optimization efficiency, as well as a more effective learning process. Experiments show that our algorithm outperforms baselines by up to 40% on complex reasoning tasks with parameters reduced up to 80% and is more robust to noisy edges on large scale graphs.

1. Introduction

The world is constructed through complex schemes and processes involving interactions between elements and objects. Being able to build models that can perform reasoning based on observations through establishing connections among elements and passing information, and make conclusions that can be generalized to new instances, has been a coveted guerdon of machine learning research. To perform such flexible reasoning among entities or variables, graph-structured models have been one of the default options to resort to (Sha & Pereira, 2003; Lafferty et al., 2001; Taskar et al., 2004). Graph Neural Networks (GNNs) emerged in recent years as the most popular deep learning models for processing relational data (Li et al., 2015; Deng et al., 2016), learning representation in graphs (You et al., 2019; Xu et al., 2018) and performing complex reasoning based on observations (Teney et al., 2017; Palm et al., 2018; Santoro et al., 2017).

Existing Graph neural networks encompass two key components: (1) message functions $F_{\boldsymbol{\theta}}$ which captures the relations and interactions among nodes; (2) a fixed information aggregation process $\boldsymbol{\tau}$ that passes information around the graph through message functions. Given input features $\boldsymbol{x}$ and targets $\boldsymbol{y}$, the model maximizes a general objective as follows with two components working cooperatively:

$$\max_{\boldsymbol{\theta}} p(\boldsymbol{y}|\boldsymbol{x}; F_{\boldsymbol{\theta}}, \boldsymbol{\tau}). \tag{1}$$

Copious works are developed towards building powerful message functions. Linear weight matrices are used for semi-supervised node classification in (Kipf & Welling, 2016). More complex non-linear message functions are adopted for high-dimensional relational reasoning (Santoro et al., 2017; Gilmer et al., 2017). To modulate information flows in GNNs, gated functions (Li et al., 2015; Deng et al., 2016) and attention schemes (Veličković et al., 2017) are applied to yield more flexible models. However, building an algorithm that systematically discusses and models the information aggregation process $\boldsymbol{\tau}$ has been under-explored.

In our paper, we propose a novel algorithm, termed *Policy Message Passing (PMP)* that generalizes existing GNNs by representing the inference process as a learnable component modeled by a neural network π_{ϕ}. Instead of relying on a repetitive and fixed aggregation procedure, our algorithm tackles the high-dimensional search space induced by all possible passing combinations across the sequential procedure, and perform inference in an active "reasoning" manner. Our algorithm treats the embeddings in graphs as

[1]Princeton University [2]Borealis AI. Correspondence to: Zhiwei Deng <zhiweid@cs.princeton.edu>.

Main part of the paper was done when Zhiwei Deng was interning at Borealis AI

states, actively queries information, and performs reasoning actions. Starting from an initial graph states $\boldsymbol{s}_0$, the algorithm builds towards a distribution-based perspective that considers all possible reasoning process and learns to generate more effective samples, resulting in a new objective to be optimized:

$$\max_{\boldsymbol{\phi},\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{\tau}\sim\pi_{\boldsymbol{\phi}}}\big[p(\boldsymbol{y}|\boldsymbol{s}_0,\boldsymbol{x},\boldsymbol{\tau};F_{\boldsymbol{\theta}})\big]. \tag{2}$$

More specifically, compared with Eqn. (1), we introduce a neural network $\pi_{\boldsymbol{\phi}}$ in Eqn. (2) with additional flexibility to represent the distribution over all possible inference trajectories in GNNs.

By taking the distribution perspective of message passing and reformulating the whole information aggregation process as stochastic sequential processes, our model is empowered with abilities to capture more aspects of the graph-structure reasoning problem: (i) *Distribution nature* - the possible inference trajectories in graph goes beyond simple synchronous aggregation, and reside in a vast space with various possibilities, where each of them can lead to a different result; (ii) *Consistent inference* - reasoning steps between nodes and across steps are not isolated, maintaining the memory of reasoning history over time could lead to more effective and consistent inference, or possible message function compositions; (iii) *Uncertainty* - inference itself encodes a prior knowledge on how to perform certain tasks, representing the inference from a distributional perspective maintains the uncertainty of the results of tasks.

Our primary contribution includes: (I) We reformulate the whole inference aggregation from the Bayesian perspective and introduce an algorithm that conducts function-set based message passing with history-aware reasoning over graphs; (II) We propose a Variational Inference based framework for learning the inference models, and provide theoretical interpretation of our model's superiority from perspectives of coding theory and optimization; (III) We explore multiple designs of our model and empirically demonstrate that having a powerful inference machine can enhance the prediction in graph-structured models while achieving better parameter efficiency, on complex reasoning tasks and large scale graphs with noisy edges.

2. Preliminaries: Graph Neural Networks

We start with introducing some notations. Let $G = \{\mathcal{V},\mathcal{E},\mathcal{X}\}$ denotes a graph with local features $\mathcal{X} = \{\boldsymbol{x}^i\}$, where $\mathcal{V} = \{\boldsymbol{v}^1,\boldsymbol{v}^2,...,\boldsymbol{v}^{|\mathcal{V}|}\}, \boldsymbol{v}_i \in \mathbb{R}^d$ is the set of node states, $\boldsymbol{E} \in \mathcal{E}$ is the graph structure given a priori and each node state $\boldsymbol{v}^i$ is associated with a local feature $\boldsymbol{x}^i$. We use $\mathcal{Y} = \{\boldsymbol{y}^i\}$ to denote the output set.

A large volume of Graph Neural Network (GNN) models, which map the input features $\mathcal{X}$ to the desired output $\mathcal{Y}$ through a sequence of local message passing on node states $\mathcal{V}$, have been developed for varied applications and tasks (Defferrard et al., 2016; Li et al., 2015; Kipf & Welling, 2016; Henaff et al., 2015; Duvenaud et al., 2015). These methods share two common structures: propagation networks and output networks. In our paper, without loss of generality, we follow a standard method that propagates local messages between every pair of nodes.

Propagation networks: Assume that a graph state $\boldsymbol{s}_0 = \big(\{\boldsymbol{v}_0^i\}\big)$ is initialized from corresponding local node features. The *propagation network*, composed of a fundamental unit termed as "graph-to-graph" module, consecutively generates a sequence of graph states $(\boldsymbol{s}_0,\boldsymbol{s}_1,...,\boldsymbol{s}_T), t \in \{0,1,...,T\}$, where each states corresponds to updated node embeddings. At each timestep t, the "graph-to-graph" module takes the current states $\boldsymbol{s}_t$ and performs message passing locally between node pairs. The functions that generate messages are implemented as neural networks. The process is executed in the following order:

$$\boldsymbol{m}_t^{(i,j)} = f_{k(i,j)}\Big(\boldsymbol{v}_t^i,\boldsymbol{v}_t^j\Big);\ \boldsymbol{v}_{t+1}^i = g\Big(\sum_{j\in\mathcal{N}_i}\boldsymbol{m}_t^{(i,j)},\boldsymbol{v}_t^i,\boldsymbol{v}_0^i\Big),$$

where $f_{k(i,j)}(\cdot,\cdot):\mathbb{R}^{2d}\to\mathbb{R}^d$ is the message function and $k(i,j):\mathbb{N}_0^2\to\mathbb{N}_0$ is the function index determined by indices of node pairs. Function $g(\cdot):\mathbb{R}^{3d}\to\mathbb{R}^d$ is the aggregator function that collects messages back to node vectors. $\mathcal{N}_i$ represents the neighbours of node i in the graph. The whole aggregation process is fixed and defined as synchronous processes in existing GNNs.

Output networks: The *output network* of the model maps the information in the graph states to the target output through function $f_o:\mathcal{V}\to\mathcal{Y}$. The mapping function is generally represented by a neural network. Note that the target variable $\boldsymbol{y}\in\mathcal{Y}$ can take various forms, such as node-level labels, graph-level labels or even sentences, depending on the tasks. Without the loss of generality, we use node-level labels in our following sections.

Limitations: The standard GNN described above represents a constrained class of models that focus on designing expressive functions capturing interactions between nodes, but ignore the modeling of the execution processes of passing messages, which is crucial in controlling and editing the node vector values. This prevents the model from being a more general class of approaches, which learn to actively reason over graph states through modeling both function designs and the aggregation process. Potential benefits of this class of model include performing edge discovery or removal dynamically, making compositional usage of message functions across reasoning steps, or learning the order of solving problems (Chen et al., 2018).

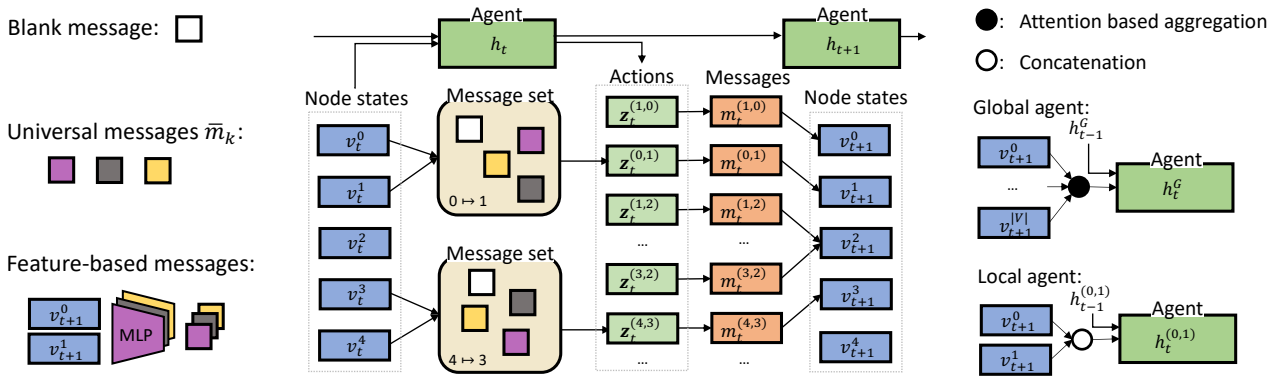

Figure 1. PMP algorithm. Our PMP algorithm consists of two components jointly trained under VI framework: (1) Function-set message passing based on actions proposed by agents; (2) an agent that propose and executes the actions on the message function sets. **Left**: two possible designs of messages. **Right**: two possible designs for agents. **Middle**: one step reasoning over the graph.

3. Our Approach: Policy Message Passing

In this section, we describe our proposed Policy Message Passing algorithm, under the same problem setup. Given the graph defined as $G = \{\mathcal{V}, \mathcal{E}, \mathcal{X}\}$ and the graph initial states $\boldsymbol{s}_0$, we build our algorithm to model the full information aggregation process (referred to as reasoning process), to generate desired output $\mathcal{Y}$. In the following, we first discuss the general algorithm framework, then describe our model designs and show how our proposed model broadens the class of standard GNNs. We also theoretically show the benefits of building a more expressive reasoning process.

Framework We formulate our algorithm from a Bayesian perspective, and treat the reasoning process as a latent variable $\boldsymbol{\tau}$. The reasoning process is defined to be a sequence of actions $\boldsymbol{\tau} = (\boldsymbol{a}_0, \boldsymbol{a}_1, ..., \boldsymbol{a}_T)$, where T is the number of reasoning steps and $\boldsymbol{a}_t$ is the action defined over the whole graph. As the message passing operation is defined locally between pairs of nodes, denoted as $\boldsymbol{z}_t^{(i,j)} \in \mathcal{Z}$, the graph-level action can then be defined by the collection of local operations $\boldsymbol{a}_t = \{\boldsymbol{z}_t^{(i,j)}\}$. As a consequence, if consider $\boldsymbol{z}$ as a dicrete variable with $\mathcal{Z}$ denoting the set of possible values, the search space for the model is $|\mathcal{Z}|^{T|V|^2}$, where $|V|$ is the number of nodes. Besides the large search space, the information passing process also has a distribution nature: there are multiple possibilities of transmitting information and manipulating node contents.

To model the distribution over the full reasoning process $\boldsymbol{\tau}$, we define three key components: (1) the set of message functions $F_{\boldsymbol{\theta}}^{(i,j)} = \{f_k(\cdot, \cdot; \boldsymbol{\theta}_k)\}_{(i,j)}$ used for passing between each pair of nodes i and j; (2) a reasoning agent $\pi_{\boldsymbol{\phi}}(\cdot; \boldsymbol{\phi})$, parameterized by $\boldsymbol{\phi}$, which determines the reasoning processes through message function selections; (3) a Variational Information framework which performs inference and learning of the reasoning processes. Although we are able to specify varied function sets based on node indices (i, j), we adopt a shared function set $F_{\boldsymbol{\theta}} = \{f_k(\cdot, \cdot; \boldsymbol{\theta}_k)\}$ across all node pair (i, j) for simplicity. Detailed discussions for each component is provided in the following sections.

3.1. Function set-based message passing

Standard Graph Neural Networks typically assign a unique message function $f_k(\cdot, \cdot)$ to each node pairs, where the k is the index of function type. The message function needs to account for node states and generate a message as an output from node i to j. To build a powerful and flexible reasoning process, we instead define a message function set $\{f_k(\cdot, \cdot; \boldsymbol{\theta}_k)\}$ to accommodate multiple possibilities between node pairs. The definition of message functions can be either driven by the prior knowledge (e.g. potential semantic relations between nodes), or separate random-initialized neural networks where message types emerge during learning process. In the following, we present two designs of message function set $F_{\boldsymbol{\theta}} = \{f_k(\cdot, \cdot; \boldsymbol{\theta}_k)\}$.

Feature-based messages Similar to standard GNNs, feature-based messages are derived from node features or states through a direct function mapping $f_k(\boldsymbol{v}_t^i, \boldsymbol{v}_t^j; \boldsymbol{\theta}_k)$. The function takes in node states at time t, and produce a message $\boldsymbol{m}_{t,k}^{(i,j)} = f_k(\boldsymbol{v}_t^i, \boldsymbol{v}_t^j; \boldsymbol{\theta}_k)$. The produced message $\boldsymbol{m}_{t,k}^{(i,j)}$ is feature aware and could vary due to subtle changes in node states. $\boldsymbol{\theta}_k$ is the parameters for function k. In our model, the function is represented by a Multi-Layer Perceptron (MLP).

Universal messages To build models that can execute the passing message operations to control node content, it might be beneficial to learn a set of universal messages, which can unload the modeling from typical non-linear functions to the agent. Inspired by the concept of Vector Quantisation technique used in distribution approximation (van den Oord et al., 2017; Louizos et al., 2017), we use a set of "universal" message vectors $\{\bar{\boldsymbol{m}}_k\}$ that represent possible

prototypes of information. The prototypical messages are later selected and transformed linearly (scale and shift): $\boldsymbol{m}_{t,k}^{(i,j)} = \alpha_{t,k}^{(i,j)} \bar{\boldsymbol{m}}^k + \beta_{t,k}^{(i,j)}$, in which the selection, scale and shift coefficients are actions generated by the reasoning agents. The message function k of $F_{\boldsymbol{\theta}} = \{f_k(\cdot,\cdot;\boldsymbol{\theta}_k)\}$ and corresponding learnable parameters, in this case, can be expressed as $f_k(i,j;\theta_k = \bar{\boldsymbol{m}}_k) = I \cdot \bar{\boldsymbol{m}}_k$.

Set-based message passing In both cases, we have a set of functions $F_{\boldsymbol{\theta}} = \{f_k(\cdot,\cdot;\boldsymbol{\theta}_k)\}, k \in \{1,...,K\}$ defined between each pair of nodes i and j. Note that in general the sets assigned for edges can vary or be manually specified. Given the message function set between node i and j, model's reasoning agents choose to perform an action $\boldsymbol{z}_t^{(i,j)}$ over the elements in the set. The final message $\boldsymbol{m}_t^{(i,j)}$ at time t passed from node i to node j is derived under the operation defined as: $\boldsymbol{m}_t^{(i,j)} = \boldsymbol{z}_t^{(i,j)}\big(F_{\boldsymbol{\theta}}\big)$, where $\boldsymbol{z}_t^{(i,j)}(\cdot)$ is the function defined by the action, e.g. indicator function based selection or linear transformation. We define the action to be a vector of operations $[\boldsymbol{z}_{t,0}^{(i,j)}, \boldsymbol{z}_{t,1}^{(i,j)}, ..., \boldsymbol{z}_{t,K}^{(i,j)}]$ on each message function type, the final messages between node i and j are derived by: $\boldsymbol{m}_t^{(i,j)} = \sum_{k=1}^{K} \boldsymbol{z}_{t,k}^{(i,j)}\big(\boldsymbol{m}_{t,k}^{(i,j)}\big)$. For example, for feature-based messages, the action is a selection operation, while for universal messages, an extra linear transformation should also be applied. For time step t, the full update of the node states will be:

$$\boldsymbol{v}_{t+1}^i = g\Big(\sum_{j\in\mathcal{N}_i}\sum_{k=1}^{K} z_{t,k}^{(i,j)}\big(f_k(\cdot,\cdot;\boldsymbol{\theta}_k)\big), \boldsymbol{v}_t^i, \boldsymbol{v}_0^i\Big),$$

where $g(\cdot)$ is a parameterized aggregation function that updates the node state by messages, previous node states and initial states. $f_k(\cdot,\cdot;\boldsymbol{\theta}_k)$ is the k-th function that generates the message $\boldsymbol{m}_{t,k}^{(i,j)}$ at each step t according to the inputs from node i and j. $\mathcal{N}_i$ is the set of neighbors of node i in the graph. Note that we also define an "null" function $f_0(\cdot,\cdot) = \mathbf{0}$ as 0-th function for all sets to allow for the choice of removing connectivity between nodes.

3.2. Execution Agents

Starting from the initial states $\boldsymbol{s}_0$, an effective information aggregation process will execute the local message passing operations to iteratively edit, transform and control the graph states, leading to a states sequence $(\boldsymbol{s}_0, \boldsymbol{s}_1, ..., \boldsymbol{s}_T)$. The graph states $\boldsymbol{s}_t$ should be informative about target $\mathcal{Y}$ and induce an easy mapping for the output networks (described later) $f_o : \mathcal{V} \to \mathcal{Y}$. We define the graph states at each time step t to be the collection of node states and message states $\boldsymbol{s}_t = \{\boldsymbol{v}_t^i\}$. To derive such a sequence of states, we build an agent that models the whole reasoning process between all pairs of nodes across the T steps. The agent tries to learn a general inductive bias of performing operations, selecting order and composing actions across reasoning steps. We define the agent to be $\pi(\boldsymbol{\tau};\boldsymbol{\phi}) = \pi(\boldsymbol{\tau}|\boldsymbol{s}_0, \boldsymbol{s}_1, ..., \boldsymbol{s}_T;\boldsymbol{\phi})$, where $\boldsymbol{\tau} = (\boldsymbol{a}_0, \boldsymbol{a}_1, ..., \boldsymbol{a}_T)$ is the sequence of actions that the agent takes. The distribution over action trajectory $\boldsymbol{\tau}$ can be further decomposed as $\prod_{t=0}^{T} q(\boldsymbol{a}_t|\boldsymbol{s}_{\leq t})$, where $q(\boldsymbol{a}_t|\boldsymbol{s}_{\leq t})$ is per-step action distributions. In the following, we discuss two designs to show how the agents can collect information from the graph and propose actions.

Global agent A global reasoning agent collects information from the whole graph. Specifically, it has access to all node states $\{\boldsymbol{v}_t^i\}$ at step t. We use a variant of recurrent neural network (RNN), namely the Gated Recurrent Unit (GRU) (Chung et al., 2014), to maintain the hidden memory over reasoning history. Denoting the hidden state at time t as $\boldsymbol{h}_t$, the agent first use a one-hop soft attention (Bahdanau et al., 2014; Deng et al., 2016) to check all the node state vectors, based on its hidden memory state $\boldsymbol{h}_t^G$. The attention weight c_t^i is calculated by $c_t^i = \exp(\bar{c}_t^i)/\sum_j \exp(\bar{c}_t^j)$, where $\bar{c}_t^i = (W_{hc}\boldsymbol{h}_t^G)^T(W_{vc}\boldsymbol{v}_t^i)$. The attended node information is used for updating hidden memory:

$$\boldsymbol{h}_t^G = \mathrm{GRU}_G\big(\textstyle\sum_i c_t^i \boldsymbol{v}_t^i, \boldsymbol{h}_{t-1}^G\big).$$

Action distributions are modeled and generated through an *Action head network*. Note that action $\boldsymbol{a}_t$ is the set of all local actions for message between each pair of nodes. The distribution $q(\boldsymbol{z}_t^{(i,j)}|\boldsymbol{v}_t^i, \boldsymbol{v}_t^j, \boldsymbol{h}_t^G)$ over local action $\boldsymbol{z}_t^{(i,j)}$ are parameterized through the output of an Multi-Layer Perceptron: $\mathrm{MLP}\big(\boldsymbol{v}_t^i, \boldsymbol{v}_t^j, \boldsymbol{h}_t^G\big)$, which could generate probability values for Categorical distributions, or mean and variance of Gaussian distributions.

Local agent The local agent-based modeling instead builds a pool of agents, in which each of them accounts for the operation of a single pair of nodes. Similar to the global agent setting, every local agent uses a GRU to maintain the memory hidden state based on the local state dynamics of the corresponding node pair. All agents share the same learnable parameters to generalize across different graph sizes. The memory hidden states are updated as follows:

$$\boldsymbol{h}_t^{(i,j)} = \mathrm{GRU}_L\big(\boldsymbol{v}_t^i, \boldsymbol{v}_t^j, \boldsymbol{h}_{t-1}^{(i,j)}\big).$$

Based on the hidden states, at every step t, the agent proposes an action distribution $q(\boldsymbol{z}_t^{(i,j)}|\boldsymbol{h}_t^{(i,j)})$. The sufficient statistic of the action distribution is generated from $\mathrm{MLP}\big(\boldsymbol{h}_t^{(i,j)}\big)$.

Action head and differentiable sampling: The action distribution $q(\boldsymbol{a}_t|\boldsymbol{s}_{\leq t})$ at each step is defined over all the local actions $\boldsymbol{a}_t = \{\boldsymbol{z}_t^{(i,j)}\}$. We use a factorized distribution $q(\boldsymbol{a}_t^{(i,j)}|\boldsymbol{s}_{\leq t}) = \prod_{i,j} q(\boldsymbol{z}_t^{(i,j)}|\boldsymbol{s}_{\leq t})$, where each of them is derived from either the global agent or the local agent. Given the distributions $q(\boldsymbol{z}_t^{(i,j)}|\boldsymbol{s}_{\leq t})$, we can now use them to perform operations by sampling actions from these distributions. In our algorithm, we define the action $\boldsymbol{z}$ according to the choice of message function set: (1) for standard

feature-based message functions, we use the Categorical distribution to represent the probabilities of selecting each type of functions; (2) for universal messages, where the action operation is the combination of function type, coefficient scale and shift, we factorize $q(\boldsymbol{z}_t^{(i,j)}|\boldsymbol{s}_{\leq t})$ as the product of Categorical distribution over function types, and two Gaussian distributions over scalar coefficients α and β. For Gaussian distributions, we use reparameterization trick (Kingma & Welling, 2013) for backpropagating the gradients. For Categorical distribution, although the sampling process of actions is straightforward, differentiating through the sampling requires reparameterization tricks on discrete variables. In our implementation, the Gumbel softmax trick (Jang et al., 2016) is used for differentiable sampling:

$$\tilde{z}_{t,k}^{(i,j)} = \frac{\exp((\log(z_{t,k}^{(i,j)}) + \epsilon_k)/\lambda)}{\sum_{m=1}^{K} \exp((\log(z_{t,m}^{(i,j)}) + \epsilon_m)/\lambda)},$$

where $z_{t,k}^{ij}$ is the probability value for each category, ϵ_k is sample drawn from the Gumbel(0,1) distribution and λ is the temperature parameter of softmax. As λ approaches 0, the distribution over $\tilde{\boldsymbol{z}}_t^{ij}$ converges to a Categorical distribution with one-hot encoding, leading to the discrete action selections for agents at each timestep. The sampled actions are used to perform set-based message passing and update node state $\boldsymbol{s}_t$ according to Sec. 3.1.

Output networks Similar to standard GNNs, we use an output network to map the graph states $\boldsymbol{v}$ to the target output variable $\boldsymbol{y}$ through $f_o : \mathcal{V} \to \mathcal{Y}$. As proposed in (Kipf et al., 2018; Palm et al., 2018; Wei et al., 2016), mapping from the graph states $\boldsymbol{s}_t$ to the target variable at every step instead of final step helps to solve the gradient vanishing issue and improve the performance. We have similar observations in our experiments and follow the same setup for training. Formally, our log-likelihood term to be optimized is:

$$\log p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\tau}) = \sum_t \log p(\boldsymbol{y}_t|\boldsymbol{s}_{\leq t}, \boldsymbol{\tau}).$$

The output network is realized by an MLP with learnable parameters. For example, in the node classification setting, the MLP generates the logits of Multinomial distibutions.

3.3. Learning objective

Given the node features $\boldsymbol{x}$, the target variable $\boldsymbol{y}$, the graph structure $\boldsymbol{E}$, and the agents $\pi(\boldsymbol{\tau}|\boldsymbol{s}, \boldsymbol{\phi})$, we formulate the model under the Variational Inference framework (Kingma & Welling, 2013): we would like to infer and approximate the distribution as close to the true posterior distribution $p(\boldsymbol{\tau}|\boldsymbol{x}, \boldsymbol{y})$ over the reasoning process $\boldsymbol{\tau}$ as possible. We use the parameterized agent π to represent a variational distribution family and minimize the KL Divergence between π and the true posterior: $p(\boldsymbol{\tau}|\boldsymbol{x}, \boldsymbol{y})$. Note that the variational distribution $\pi(\boldsymbol{\tau}|\boldsymbol{s}, \boldsymbol{\phi})$ doesn't have access to the ground truth variable $\boldsymbol{y}$, which enforce it to learn the general inductive bias of reasoning over graph only based on the input feature and graph states. By re-writing the term $KL\Big(\pi(\boldsymbol{\tau}|\boldsymbol{s})||p(\boldsymbol{\tau}|\boldsymbol{x}, \boldsymbol{y})\Big)$, we derive the following equations:

$$\begin{aligned}
&\log p(\boldsymbol{y}|\boldsymbol{x}) - KL\Big(\pi(\boldsymbol{\tau}|\boldsymbol{s})||p(\boldsymbol{\tau}|\boldsymbol{x}, \boldsymbol{y})\Big) \\
&= \mathbb{E}_{\boldsymbol{\tau}\sim\pi}\Big[\log p(\boldsymbol{y}|\boldsymbol{\tau}, \boldsymbol{s})\Big] - KL\Big(\pi(\boldsymbol{\tau}|\boldsymbol{s})||p(\boldsymbol{\tau})\Big) \\
&= \mathbb{E}_{\boldsymbol{\tau}\sim\pi}\Big[\log p(\boldsymbol{y}|\boldsymbol{\tau}, \boldsymbol{s})\Big] - KL\Big(\prod_t q(\boldsymbol{a}_t|\boldsymbol{s}_{\leq t})||\prod_t p(\boldsymbol{a}_t)\Big).
\end{aligned}$$

In the above equation, the prior distribution over reasoning process $\boldsymbol{\tau}$ can be derived from the graph structure or manually specified. The decomposed KL term between $q(\boldsymbol{a}_t|\boldsymbol{s}_{\leq t})$ and $p(\boldsymbol{a}_t)$ are approximated by Monte Carlo sampling as the reasoning process progresses. Since KL Divergence is non-negative, the right hand side of the equation is known as *Evidence Loader Bound (ELBO)*. During training, for every graph data point $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{E})$, we initialize the graph states as $\boldsymbol{s}_0$ from $\boldsymbol{x}$, use the agent to sample actions from produced the action distributions, execute the action through the set-based message passing, and map the graph states to the target $\boldsymbol{y}$ through the output network to maximize the likelihood. The proposed action distribution is regularized by the prior distribution induced from graph structure $\boldsymbol{E}$.

4. Discussions and Theoretical insight of PMP

In this section, we show that our PMP algorithm generalizes the existing GNNs by modeling the information aggregation process as a stochastic reasoning process. The resulting PMP based GNNs can potentially lead to a more effective decoding process and more efficient optimization.

4.1. Coding theory perspective

It is well-known that, from an information-theoretic view, there is a close connection between Variational Inference, Bits-Back Coding (Hinton & Van Camp, 1993; Honkela & Valpola, 2004), and Minimum Description Length (MDL) (Rissanen, 1978). In the following, we show that the advantage of our algorithm can be interpreted from the perspective of coding theory.

Let $\boldsymbol{x}$ be the node features for a graph, $\boldsymbol{y}$ be node-level labels, and $\boldsymbol{\tau}$ be the inference/aggregation trajectories of graph neural networks. As $\boldsymbol{\tau}$ is not observed, we treat it as a latent variable to be inferred using Variational Inference framework. Assume that we have an encoder $q(\boldsymbol{\tau}|\boldsymbol{x}, \boldsymbol{\phi})$, a standard Variational Inference framework optimizes (mini-

mize) the following quantity:

$$-\mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim data}\Big[\mathbb{E}_{\boldsymbol{\tau}\sim q(\boldsymbol{\tau}|\boldsymbol{x};\boldsymbol{\phi})}\big[\log p(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{\tau};\boldsymbol{\theta}) \tag{3}$$

$$+\log p(\boldsymbol{\tau})-\log q(\boldsymbol{\tau}|\boldsymbol{x};\boldsymbol{\phi})\big]\Big] \tag{4}$$

From the coding perspective, if we view the above equation as a coding process, the coding cost contains two parts: $C = C_{rec} + C_{coding}$ to be optimzied, where C_{rec} corresponds to likelihood term $\log p(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{\tau};\boldsymbol{\theta})$ in (3) and C_{coding} is the coding cost with "Bits-Back" part subtracted in (4). Note that during testing, we only care about the first part of the coding.

We use $q(\boldsymbol{\tau};\boldsymbol{\phi})$ to represent a general encoder. The reconstruction coding cost C_{rec} could be re-written as:

$$\begin{aligned}C_{rec} =& -\mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim data}\Big[\log p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})\Big]\\ &-\mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim data}\Big[\mathbb{E}_{\boldsymbol{\tau}\sim q(\boldsymbol{\tau};\boldsymbol{\phi})}\big[(p(\boldsymbol{\tau}|\boldsymbol{x},\boldsymbol{y},\boldsymbol{\theta})-\log p(\boldsymbol{\tau}))\big]\Big]\end{aligned}$$

where $p(\boldsymbol{\tau}|\boldsymbol{x},\boldsymbol{y},\boldsymbol{\theta})$ and $p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})$ are posterior distribution over $\boldsymbol{\tau}$ and likelihood term under parameters $\boldsymbol{\theta}$. If we set $q(\cdot)$ as the optimal distribution which is the true posterior distribution under $\boldsymbol{\theta}$, then we have

$$\begin{aligned}C^1_{rec} =& -\mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim data}\Big[\log p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})\Big]\\ &-\mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim data}\Big[KL(p(\boldsymbol{\tau}|\boldsymbol{x},\boldsymbol{y},\boldsymbol{\theta})||p(\boldsymbol{\tau}))\Big].\end{aligned}$$

Instead, if we set $q(\cdot;\boldsymbol{\phi})$ as the prior distribution over $\boldsymbol{\tau}$ (i.e., we don't provide any encoding information from $\boldsymbol{\tau}$ side), then we have

$$\begin{aligned}C^2_{rec} =& -\mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim data}\Big[\log p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})\Big]\\ &+\mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim data}\Big[KL(p(\boldsymbol{\tau})|p(\boldsymbol{\tau}|\boldsymbol{x},\boldsymbol{y},\boldsymbol{\theta}))\Big].\end{aligned}$$

This implies $C^1_{rec} \leq C^2_{rec}$, where the equality holds when the posterior distribution $p(\boldsymbol{\tau}|\boldsymbol{x},\boldsymbol{y})$ equals the prior distribution $p(\boldsymbol{\tau})$, which means the best aggregation process $\boldsymbol{\tau}$ is to assume nothing (this can sometimes appear in too simple tasks/applications). In harder cases where GNN is trying to tackle a highly complex task, we can see that if the distribution is close to the true posterior, it can always help to get a shorter length of coding over the reconstruction term. This is exactly what variational inference is optimizing, which minimizes the distance between a distribution $q(\boldsymbol{\tau}|\boldsymbol{x};\boldsymbol{\theta})$ and the true posterior distribution $p(\boldsymbol{\tau}|\boldsymbol{x},\boldsymbol{y};\boldsymbol{\phi})$.

4.2. Optimization perspective

If we treat the parameters from $q(\boldsymbol{\tau}|\boldsymbol{x},\boldsymbol{y},\boldsymbol{\theta})$ and $p(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{\tau};\boldsymbol{\phi})$ as two distinct sets of parameters, then $\boldsymbol{\theta}$ controls the generation of reasoning process, while $\boldsymbol{\phi}$ controls the likelihood model. We demonstrate that: (1) through considering actions (i.e., indicator functions in GNNs), we have a stronger learning model; (2) if one set of parameters is more sensitive, it's computationally more efficient to directly optimize on them.

For the optimization problem, we consider fixing the parameter space for $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ when maximizing the likelihood:

$$L(\boldsymbol{\theta},\boldsymbol{\phi}) = \mathbb{E}_{\boldsymbol{\tau}\sim q(\boldsymbol{\tau}|\boldsymbol{x},\boldsymbol{y},\boldsymbol{\phi})}\Big[p(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{\tau},\boldsymbol{\theta})\Big]$$

On the other hand, when the parameter spaces for $\boldsymbol{\phi}$ are inclusive, i.e. $\boldsymbol{\Psi}_1 \subset \boldsymbol{\Psi}_2$, then we have

$$\max_{\boldsymbol{\phi}\in\boldsymbol{\Psi}_1,\boldsymbol{\theta}\in\boldsymbol{\Theta}} L(\boldsymbol{\theta},\boldsymbol{\phi}) \leq \max_{\boldsymbol{\phi}\in\boldsymbol{\Psi}_2,\boldsymbol{\theta}\in\boldsymbol{\Theta}} L(\boldsymbol{\theta},\boldsymbol{\phi}).$$

Here $\boldsymbol{\Psi}_1$ corresponds to the existing GNNs and $\boldsymbol{\Psi}_2$ corresponds to our PMP based GNNs. This guarantees that our learning procedure can result in a lower risk due to the more expressive modeling capability.

From the perspective of parameter sensitivities, we can also achieve that a significantly smaller number of parameters are required in optimization. This is because the parameters in $q(\boldsymbol{\tau}|\boldsymbol{x},\boldsymbol{y},\boldsymbol{\phi})$ and $p(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{\tau};\boldsymbol{\theta})$ have different sensitivities to the final model performance. If we don't have a flexible $q(\cdot)$ to be learned, the burden of reconstruction all falls onto the first term $p(\cdot)$, which is modeled by the GNN architecture. When we slightly increase the number of parameters in the sensitive parameter set ($q(\cdot)$ here), we may significantly reduce the number of parameters for the less sensitive parameter set ($p(\cdot)$ here). This can significantly reduce the computation complexity for tuning parameters. We empirically show that by adding the $q(\cdot;\phi)$ distribution, we can reduce the overall number of parameters by $> 80\%$ and maintain, even increase, the performance.

4.3. Compare with GNN variants

Existing GNN variants provide two remedies to the basic GNN architecture: (1) the first class of models use gates or attentions to control the information during propagation (Veličković et al., 2017; Li et al., 2015; Deng et al., 2016). Note that these methods, although inspiring, are still constrained to non-probabilistic, single-step operations, which don't consider the full search space of the sequential process and lack systematic interpretations. (2) Several GNN variants also take a Bayesian perspective (Kipf et al., 2018; Zhang et al., 2019; Ng et al., 2018). In NRI (Kipf et al., 2018), an encoder is used to approximate the posterior distribution over graph structures, the sample of which still leads to fixed inference processes. (Zhang et al., 2019) focuses on distribution over graph convolution weights, while (Ng et al., 2018) uses Gaussian Process to model data points correlations. Both are orthogonal to our approach and could potentially benefit from our algorithm.

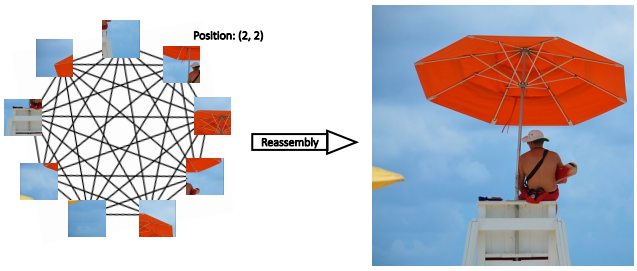

Figure 2. Illustration of the image puzzle reassembly task. An image is first split into patches which are then shuffled. A fully-connected graph is built to connect all patches. Given the position of one patch, the task requires model to infer positions for the rest.

5. Experiments

We test our proposed model on two tasks with different modalities of data: the complex visual reasoning task and the node classification on citation network benchmarks. Visual data resides in a high-dimensional space, leading to rich and diverse interactions when organized in graphs. Citation network data has fewer dimensions per node, but larger scales on the size of graphs.

5.1. Visual Reasoning — Image Puzzle Reassembly

Image puzzle reassembly is a classic problem that requires understanding complex patterns or pattern combinations in images, linking possible pairs of tiles, correcting errors and iteratively solving the puzzle. The images are firstly segmented into tiles with shuffled orders, see fig. 2. The task is to reassemble the tiles into original full images with correct positions and orders.

5.1.1. Datasets and Experimental Setup

Datasets We generate image puzzles from two large-scale benchmark database: (1) Visual Genome (Krishna et al., 2017) - The Visual Genome (VG) database is known for images with complex object layouts or interactions, and is standard benchmark data for scene graph generation task. It contains 64346 images for training and 43903 images for testing. (2) COCO (Lin et al., 2014) - We use the COCO dataset (2014 version) that contains 82783 images for training and 40504 images for testing. COCO is also standard dataset used for "recognition in context" and contains diverse objects and semantic information.

Task setup To generate a $d \times d$ puzzle from an image with size $H \times W$, we divide the image into $d \times d$ non-overlapping patches. Each patch has size $H/d \times W/d$. The d^2 patches are then shuffled into a set with size d^2. In our experiments, we first resize the images into 256×256 before generating puzzles. Three puzzle reassembly tasks are set up: $3 \times 3, 4 \times 4$, and 6×6, leading to graphs with size 9, 16 and 36 respectively. See figure 2 for illustration. The inference model will be given patch features and the position for one patch, and classify the rest. We use ResNet-18 (He et al., 2016) as our base network architecture for extracting features. The base network is jointly trained from scratch.

Models	Accuracy(%) ↑ / Kendall-tau ↑			
	2x2	3x3	4x4	5x5
(Mena et al., 2018)	1.0 / 1.0	.97 / .96	.9 / .88	.79 / .78
PMP_L - Feature	**1.0 / 1.0**	**1.0 / 1.0**	**.99 / .99**	**.99 / .99**

Table 1. We compare our model's performance with state-of-the-art deep learning based puzzle reassembly model on Celeba dataset.

5.1.2. Baselines and Implementation Details

We compare our model with four powerful graph-structured models as baselines: (1) *Graph Neural Network (GNN)* — we use the GNN described in Sec.2 as our basic baseline model to compare with. This GNN corresponds to our model without the PMP components and uses a deterministic informagion aggregation process. Models with similar architectures are widely used in other applications (Gilmer et al., 2017; Xu et al., 2017). (2) *Attention-based Graph Neural Network* (Veličković et al., 2017; Wang et al., 2018) — the attention-based GNN can be viewed as a special case of our model, with single step soft message selections. (3) *Recurrent Relational Network (RRN)* (Palm et al., 2018) — RRN is a GNN model that performs long-term reasoning via maintaining hidden memory on the node states. (4) *Neural Relational Inference model (NRI)* (Kipf et al., 2018) — the NRI model is the closest one to our model. It aims at inferring the graph structure from the data and is formulated under the Variational Inferece framework. However, the aggregation process is still fixed based on sampled structures.

Implementation details We randomly sample 30% images from training set as validation set and search hyperparameters. Numbers reported in tables are for test set. For node dimension, we found 200 can achieve maximum performance for baselines, while only 50 dims are needed for our PMP models. Number of functions is set to 4 for GNN-Attention, NRI and $\text{PMP}_{L/G}$-Feature and 32 for $\text{PMP}_{L/G}$-Universal. We use $d + 2$ inference steps for all models, where d is square root of graph size. Learning rate is set to $5e^{-4}$. Models are trained for 60 epochs. More architecture details and number of parameters per model can be found in Appendix.

5.1.3. Experimental Results

Results for image puzzle reassembly on two benchmark datasets are summarized in Table 2. Performance are measured by per node (patch) classification accuracy and correlation score Kendall-tau. We find that our PMP-based models can achieve $> 90\%$ on classification accuracy, and outperform the GNN without PMP by up to 40%. Local agent PMP overall performs better than global agent ones,

Models	Visual Genome						COCO					
	Accuracy(%) ↑			Kendall-tau ↑			Accuracy(%) ↑			Kendall-tau ↑		
	3x3	4x4	6x6	3x3	4x4	6x6	3x3	4x4	6x6	3x3	4x4	6x6
random	11.11	6.25	2.78	-	-	-	11.11	6.25	2.78	-	-	-
GNN	88.76	76.18	56.11	90.32	82.56	75.07	90.46	79.94	40.50	91.47	84.75	68.18
GNN-attention	90.96	82.05	72.25	92.12	86.75	84.66	93.75	82.77	79.36	93.55	86.75	84.66
RRN	90.83	83.03	62.66	92.48	89.51	82.01	91.51	81.26	48.88	92.76	86.24	73.92
NRI	92.31	82.21	59.90	93.00	86.78	78.15	93.91	82.25	55.51	94.73	86.97	75.49
PMP_G - Feature	93.31	89.18	84.10	94.95	93.91	89.77	96.82	88.52	80.82	96.56	93.50	90.18
PMP_G - Universal	94.26	88.62	78.23	95.28	93.56	85.84	94.11	89.16	80.16	95.44	93.19	89.29
PMP_L - Feature	94.40	**96.55**	91.40	94.44	**97.22**	95.33	**96.68**	**97.05**	91.06	96.65	**97.69**	94.59
PMP_L - Universal	**95.31**	94.55	**96.88**	**96.01**	96.64	**98.27**	96.01	94.84	**93.23**	**96.90**	95.99	**95.81**

Table 2. Performance results on Visual Genome and COCO datasets, measured by per-patch classification accuracy and correlation score Kendall-tau. PMP_L and PMP_G correspond to local agent version and global agent version respectively. "Feature" represents feature-based message set passing and "Universal" represents universal message set passing. Our model achieve up to 40% improvement over baselines.

indicating that memorizing actions and history for the whole graph is harder than factorizing the modeling into local ones. Actually local agent PMP has higher parameter efficiency too. We observe that universal messages can achieve similar (or higher) results compared to feature-based messages, showing that the agent is capable of learning a set of feature-agnostic and task-specific messages to perform reasoning.

Message function analysis *Number of message functions (NMF)* is highly correlated to the expressive power and flexibility of PMP models. With higher NMF, the PMP model. We find that, with feature-based message functions, the model achieves state-of-the-art through four different function types. For PMP with universal messages, the algorithm needs more "prototypical" vectors (32 in the experiments) to cover possible component in the message space. We also observe that universal message based models have more stable training process. This is potentially due to that the message set is disentangled from high-dimensional image features and is less sensitive to perturbations. We also find that *message function composition* appears across the consecutive operations. For example, we measure the composition rate of two function types through $\frac{\#freq(k_t=i,k_{t+1}=j)}{(\#freq(k=i)+\#freq(k=j))}$ and find the metric ranges from 0.0522 to 0.1804, indicating the usage of compound messages for reasoning.

Models	Noisy edges		
	50%	100%	200%
GCN	26.55 ± 1.3	23.53 ± 1.44	22.62 ± 1.84
GAT	58.83 ± 1.35	45.82 ± 2.55	34.53 ± 0.45
PMP_L-F	$\mathbf{65.37 \pm 1.36}$	$\mathbf{61.42 \pm 1.31}$	$\mathbf{50.3 \pm 1.13}$

Table 3. The performance of node classification for Cora dataset under varied noisy edge settings. The experiment follows the standard transductive setup in (Yang et al., 2016).

Parameter efficiency Through modeling the reasoning process $\boldsymbol{\tau}$, we find that PMP largely reduces the burden on GNNs. Specifically, a basic GNN requires 321,0809 parameters, through modeling the reasoning process with an extra 6,6592 parameters using PMP, the number of parameters for GNN can be reduced by 5 times smaller to 54,7709, resulting in a model with 18.44% of original model size.

Compare with state-of-the-art method: We compared our method with state-of-the-art deep learning based method for image puzzle assembly task on CelebA dataset. The results are shown in Table 1. Our model shows near perfect puzzle reassembly results across four settings.

5.2. Node classification with noisy edges

We further inspect on models' ability on handling noisy edges on large scale graphs. Current graph neural network benchmark datasets on citation networks, such as Cora (Sen et al., 2008), heavily rely on well caliberated edge matrix as a strong prior information for inference. The burden on aggregation process are lessened, compared to more general cases where adjacency matrix information are often noisy or missing. In this experiment, we compare our feature-based PMP with local agents to two state-of-the-art graph-structured models: (1) Graph Convolutional Networks (GCN) and (2) Graph Attention Networks (GAT) on a standard citation network benchmark Cora. Instead of using only the well designed adjacency matrix information, we generate perturbation to the matrix by adding $\{\%50, \%100, \%200\} * |E|$ more noisy edges to test the robustness of algorithms w.r.t. edge noises, $|E|$ is the number of edges in the original graph. For implementation, we use the official release of the codes on two baseline models and default settings. In our PMP model, we follow the same hyper-parameters in baselines and uses recurrent network agent with 32 hidden dimensions. Results summarized in table 3 show that our model is more robust to noisy edges in performing inference over large scale graphs. Note that as we are building local agents for each pair of nodes, generalizing to even larger scale graphs will lead to high memory costs. Sampling-based methods could potentially help to solve this problem. We leave it as future work.

6. Conclusion

We propose PMP and formulate the reasonings over graphs as learnable processes under Variational Inference framework. We show that our algorithm consistently outperforms baseline GNN variants while maintaining high parameter efficiency.

References

Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Chen, S.-F., Chen, Y.-C., Yeh, C.-K., and Wang, Y.-C. F. Order-free rnn with visual attention for multi-label classification. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pp. 3844–3852, 2016.

Deng, Z., Vahdat, A., Hu, H., and Mori, G. Structure inference machines: Recurrent neural networks for analyzing relations in group activity recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4772–4781, 2016.

Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pp. 2224–2232, 2015.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1263–1272. JMLR. org, 2017.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

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

Hinton, G. E. and Van Camp, D. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, pp. 5–13, 1993.

Honkela, A. and Valpola, H. Variational learning and bits-back coding: an information-theoretic view to bayesian learning. *IEEE transactions on Neural Networks*, 15(4): 800–810, 2004.

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

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Kipf, T., Fetaya, E., Wang, K.-C., Welling, M., and Zemel, R. Neural relational inference for interacting systems. *arXiv preprint arXiv:1802.04687*, 2018.

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

Krishna, R., Zhu, Y., Groth, O., Johnson, J., Hata, K., Kravitz, J., Chen, S., Kalantidis, Y., Li, L.-J., Shamma, D. A., et al. Visual genome: Connecting language and vision using crowdsourced dense image annotations. *International Journal of Computer Vision*, 123(1):32–73, 2017.

Lafferty, J., McCallum, A., and Pereira, F. C. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

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

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. Microsoft coco: Common objects in context. In *European conference on computer vision*, pp. 740–755. Springer, 2014.

Louizos, C., Ullrich, K., and Welling, M. Bayesian compression for deep learning. In *Advances in Neural Information Processing Systems*, pp. 3288–3298, 2017.

Mena, G., Belanger, D., Linderman, S., and Snoek, J. Learning latent permutations with gumbel-sinkhorn networks. *arXiv preprint arXiv:1802.08665*, 2018.

Ng, Y. C., Colombo, N., and Silva, R. Bayesian semi-supervised learning with graph gaussian processes. In *Advances in Neural Information Processing Systems*, pp. 1683–1694, 2018.

Palm, R., Paquet, U., and Winther, O. Recurrent relational networks. In *Advances in Neural Information Processing Systems*, pp. 3368–3378, 2018.

Rissanen, J. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

Santoro, A., Raposo, D., Barrett, D. G., Malinowski, M., Pascanu, R., Battaglia, P., and Lillicrap, T. A simple neural network module for relational reasoning. In *Advances in neural information processing systems*, pp. 4967–4976, 2017.

Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., and Eliassi-Rad, T. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

Sha, F. and Pereira, F. Shallow parsing with conditional random fields. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pp. 134–141. Association for Computational Linguistics, 2003.

Taskar, B., Guestrin, C., and Koller, D. Max-margin markov networks. In *Advances in neural information processing systems*, pp. 25–32, 2004.

Teney, D., Liu, L., and van Den Hengel, A. Graph-structured representations for visual question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9, 2017.

van den Oord, A., Vinyals, O., et al. Neural discrete representation learning. In *Advances in Neural Information Processing Systems*, pp. 6306–6315, 2017.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Wang, X., Girshick, R., Gupta, A., and He, K. Non-local neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7794–7803, 2018.

Wei, S.-E., Ramakrishna, V., Kanade, T., and Sheikh, Y. Convolutional pose machines. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 4724–4732, 2016.

Xu, D., Zhu, Y., Choy, C. B., and Fei-Fei, L. Scene graph generation by iterative message passing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5410–5419, 2017.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

Yang, Z., Cohen, W. W., and Salakhutdinov, R. Revisiting semi-supervised learning with graph embeddings. *arXiv preprint arXiv:1603.08861*, 2016.

You, J., Ying, R., and Leskovec, J. Position-aware graph neural networks. *arXiv preprint arXiv:1906.04817*, 2019.

Zhang, Y., Pal, S., Coates, M., and Ustebay, D. Bayesian graph convolutional neural networks for semi-supervised classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 5829–5836, 2019.