
Approximate inference with Graph Neural Networks

Lingxiao Zhao (lingxia1)^{*1} Ksenia Korovina (kkorovin)^{*1} Wenwen Si (wenwens)^{*1}
Mark Cheung (markcheu)^{*1}

Abstract

Probabilistic graphical models are useful for discovering and analyzing structures of many real-world applications. Belief propagation and other message-passing algorithms have traditionally been used to disseminate evidence among the nodes in the graph. However, these algorithms may be inefficient for large or loopy graphs. Here, we expand the work of Yoon et al. (2018), which uses graph neural network (GNN) as an inference machine for small-scale networks. We reproduce their results and explore how to generalize GNN to larger graphs with approximate labeling schemes and custom training procedures.

1. Introduction

Many complex real-world problems can be formulated in terms of probabilistic models, and the framework of probabilistic graphical models (PGM) assists in both the modeling and inference tasks. Graphical models provide a helpful bridge between probability and graphs, transforming a probabilistic inference task into operations on a graph. However, exact inference in graphical models is NP-hard, and hence many approximate inference algorithms have been proposed to tackle this problem, such as belief propagation, variational inference, and MCMC.

With the growing interest in deep learning, researchers start to put effort into combining deep learning and PGM with techniques such as unrolling inference of PGM using deep learning (Zheng et al., 2015). In the meantime, a new set of neural networks working on graphs, graph neural networks (GNN), shows its ability of handling structured data such as social networks, chemical molecules, texts, physical systems, and images. A general framework that summarizes most existing GNNs is proposed in Gilmer et al. (2017), which shows that a spatial message passing procedure is the main component of GNN.

Based on the evidence that GNN and PGM share many

common features, it is natural to conceive that the two directions of work can be related. However, so far, graphical models and graph neural networks communities have not established common ground. In this project, we aim to explore the applications of graph neural networks, starting with a recent paper (Yoon et al., 2018) that performs small-scale inference using a GNN, and trying different ideas to make it more scalable and powerful.

2. Background & Related Work

PGM and approximate inference. Probabilistic graphical models provide a efficient statistical framework for modeling conditional independencies between random variables. However, performing inference tasks such as computing marginals or maximum a posteriori estimate is intractable in general. Hence, in practice, they are approximated by techniques like belief propagation, MCMC sampling, and variational inference (Koller & Friedman, 2009).

Graph Neural Network. Given a graph, a recently proposed general framework called Graph Neural Network (GNN) is a way to do end-to-end feature engineering over the graph. GNN is developed based on traditional CNN, graph signal processing, and message passing/propagation. A GNN is defined as a two-component structure: a propagation layer and an aggregation layer. These two layers combine together to shape a single GNN layer, which has the ability to transform node features and edge features. Like other neural network architectures, GNN builds an end-to-end projection by using multiple GNN layers. One single GNN layer can be written as follows:

$$\mathbf{h}_{(i,j)}^l = f_{v \rightarrow e}^l(\mathbf{h}_i^l, \mathbf{h}_j^l, w_{ij}) \quad (\text{propagation}) \quad (1)$$

$$\mathbf{h}_j^l = f_{e \rightarrow v}^l(\{\mathbf{h}_{(i,j)}^l | i \in \mathcal{N}_j\}, \mathbf{x}_i) \quad (\text{aggregation}) \quad (2)$$

Where l is the layer number, $f_{v \rightarrow e}^l$ and $f_{e \rightarrow v}^l$ are parametrized function such like neural network, $\mathbf{h}_{(i,j)}^l$ is the propagated information on each edge, and \mathbf{h}_j^l is the aggregated information for each node. Structured input processing models within the GNN framework have shown excellent performance in a wide range of tasks, such as graph embedding (Hamilton et al., 2017), semi-supervised learning (Kipf & Welling, 2016), relational learning (Schlichtkrull et al., 2018), and molecular classification (Gilmer et al.,

^{*}Equal contribution ¹Carnegie Mellon University.

2017). For a comprehensive review of GNN, one can refer to Bronstein et al. (2017); Battaglia et al. (2018).

Approximate inference + NNs/GNNs There are several lines of research to efficiently solve inference problems in PGM by going beyond traditional approximate inference algorithms. First, several works show that using an **inference machine** to replace the inference procedure in VI could directly replace PGM and achieve similar or even better performance. Ross et al. (2011) proposes to train a series of predictors to mimic the message-passing procedure of belief propagation and successfully uses it in point cloud classification problem, achieving better performance than mean-field inference. Heess et al. (2013) trains a neural network to “predict” the outgoing messages based on incoming messages of belief propagation. Ramakrishna et al. (2014) uses inference machine to replace graphical model and achieves significant improvement in pose estimation task. Lin et al. (2015) builds a convolutional neural network to estimate the messages in message passing inference for CRF. Another line of work focuses on unrolling optimization procedure of variational inference using neural network. There are many contributions in this direction, and the most well known is Zheng et al. (2015) that uses an RNN to replace mean-field VI with fully-connected CRF.

Yoon et al. (2018) follows the first line that learns an inference machine using graph neural network (specifically the paper uses Gated Graph Neural Network). Two kinds of mappings of belief propagation to GNN are proposed in the paper: *msg-GNN*, where each node in the GNN corresponds to a message in belief propagation; and *node-GNN*, where each node maps to a variable in the graphical model.

The total cross-entropy loss $L(\mathbf{p}, \hat{\mathbf{p}}) = -\sum_i q_i \log \hat{p}_i(x_i)$ is minimized between the exact target ($q_i = p_i(x_i)$ for marginals or $q_i = \delta_{x_i, x_{i^*}}$ for MAP) and the GNN estimates $\hat{p}_i(x_i)$. Experiments focus on small scale binary Markov random field. Results show GNNs-based inference substantially outperform belief propagation on loopy graphs. The algorithm is able to generalize out of the training set to larger graphs and graphs with different structure.

3. Proposed method

Although the aggregation operator of GNN works similarly as the sum-product computational procedure in (loopy) belief propagation algorithm, the parameters of GNN needed to be learned in order to do a good approximate inference over graphical model. To learn a GNN model as inference machine, the first step is to generate a large enough dataset where each data point is a combination of a graphical model (we use MRF) as input, and marginal distributions for all variables as target. As shown in figure 1, we create lots of MRFs with varying number of variables and different

factor parameters, then for each MRF, we use some inference algorithms to get marginal distribution of each variable. Notice that this is the main bottleneck inside our pipeline, since getting inference results for large graphical models is extremely time-consuming. For small graphical models, i.e. the number of variables is less than 20, we could get inference labels fast. Once we get a large enough dataset, we could train the inference machine whose input is a graphical model and output is marginal distributions. The training procedure is the same as any neural network based supervised learning that minimize the distance between model output and target by using first-order optimization method such as SGD and Adam. In next section we talk about how to design an inference machine based on GNN.

3.1. Baseline: Gated GNN as an Inference Machine (Yoon et al., 2018)

FROM GRAPHICAL MODEL TO GNN

As Yoon et al. (2018), we focus on inference over binary Markov Random Field (MRF). Let $\mathbf{x} \in \{-1, +1\}^{|\mathcal{V}|}$ be the binary variables in MRF where \mathcal{V} is the index set of variables. Then the the joint probability of \mathbf{x} is:

$$p(\mathbf{x}) = \frac{1}{Z} \exp(\mathbf{b}^T \mathbf{x} + \mathbf{x}^T \mathbf{J} \mathbf{x}) = \frac{1}{Z} \prod_{i \in \mathcal{V}} \phi_i(x_i) \prod_{i,j \in \mathcal{V}} \phi_{ij}(x_i, x_j) \quad (3)$$

where $\phi_i(x_i) = e^{b_i x_i}$ and $\phi_{ij}(x_i, x_j) = e^{J_{ij} x_i x_j}$ are factors in MRF with parameters \mathbf{b} and \mathbf{J} .

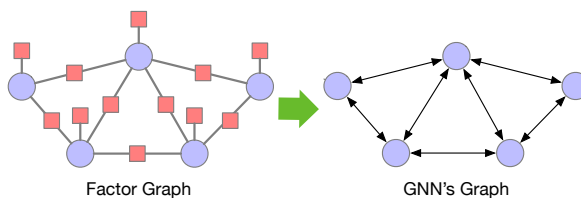


Figure 2. Mapping MRF to graph of GNN

As shown in figure 2, we can map MRF to the graph inside GNN. To be more specific, we map variables in MRF to nodes in graph of GNN, and map pairwise dependency of any two variables in MRF to edge in GNN graph. After mapping, we get a graph $G = (\mathcal{V}, \mathcal{E})$ whose edge $e_{ij} \in \mathcal{E}$ exists if and only if $J_{ij} \neq 0$, which can be used by GNN model. What’s more, MRF’s parameters \mathbf{b} and \mathbf{J} , which characterize the joint distribution of MRF, becomes input to GNN model.

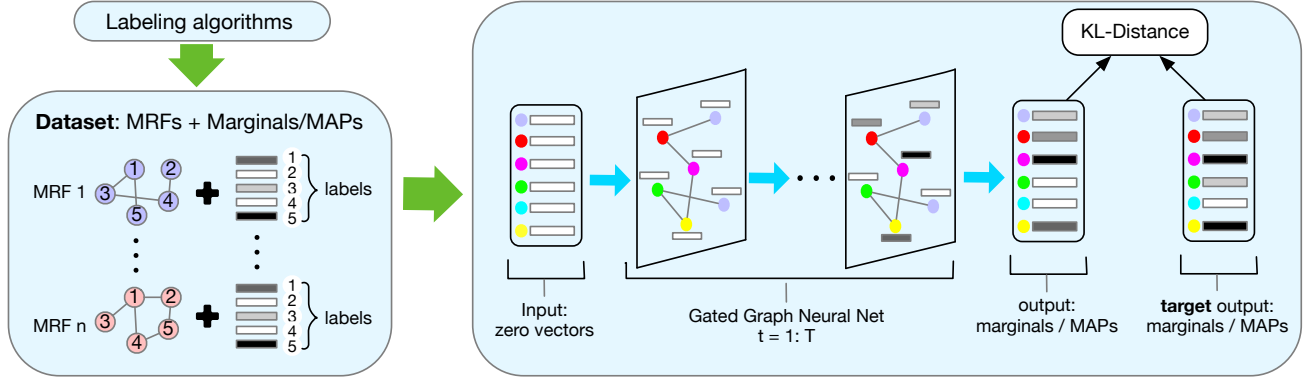


Figure 1. Pipeline for proposed GNN based approximate inference

DETAILS OF GATED GNN

Let $\mathbf{e}_{ij} = [J_{ij}, b_i, b_j]$ be the feature of edge e_{ij} , and $\mathbf{h}_i^t \in \mathbb{R}^D$ be the states of node i at t -th time step inside the GGNN model. We initialize this hidden state to all zeros, $\mathbf{h}_i^0 = \mathbf{0}$. At each step t , we update the edge messages $\mathbf{m}_{i \rightarrow j}^{t+1} \in \mathbb{R}^P$ by

$$\mathbf{m}_{i \rightarrow j}^{t+1} = \text{MLP}(\mathbf{h}_i^t, \mathbf{h}_j^t, \mathbf{e}_{ij}), \quad (4)$$

where MLP is a multilayer perceptron with ReLU activation function. We then calculate neighborhood information for every node by aggregate all incoming messages:

$$\mathbf{m}_i^{t+1} = \sum_{j \in N_i} \mathbf{m}_{j \rightarrow i}^{t+1}. \quad (5)$$

To update hidden state at $t+1$ for every node, we use a gated recurrent unit (GRU) that takes t -time hidden state and aggregated neighborhood information as input:

$$\mathbf{h}_i^{t+1} = \text{GRU}(\mathbf{h}_i^t, \mathbf{m}_i^{t+1}) \quad (6)$$

After T time steps of sending messages, aggregating messages, and updating new states, we get the final states \mathbf{h}_i^T with dimensionality D . Another MLP σ with sigmoid activation function as readout function will be used to map it to final target \mathbf{y}_i of dimensionality 2 for binary MRF. More formally,

$$\mathbf{y}_i = \sigma(\mathbf{h}_i^T) \quad (7)$$

The training of the GNN model is based on minimizing the KL divergence of prediction \mathbf{y} and target \mathbf{y}_{true} .

3.2. Approximate Labeling for GNN Training

After assessing out-of-the-box scalability of the existing approach by using a model trained on small dataset, we will try to address the primary computational issue of GNN-based

method: a need for training on ground truth labels, which are infeasible to obtain even for graphs of moderate size. For that we propose three ideas described in this section.

LABEL PROPAGATION

Label Propagation is a semi-supervised algorithm that assigns labels to previously unlabeled data points. As the original implementation in `sklearn` of this method only works for graphs with non-negative edges, we devise a modification that allows us to have a properly normalized distribution after every step of propagation. At the start, we label several random subgraphs (of user-provided sizes) with exact inference. These labels are propagated to new nodes for T steps:

$$\hat{p}_i(v_i=1) \leftarrow \sigma\left(\sum_j w_{j,i} \hat{p}_j(v_j = \text{sign}(w_{j,i}))\right) \quad (8)$$

Compared to the original label propagation, which updates as $\hat{p}_i(v_i = v) \propto \sum_j w_{j,i} \hat{p}_j(v_j = v)$, $v \in \{-1, +1\}$, we now do propagation of probabilities of $+1$ and -1 over separate channels (positive and negative edges respectively), and coalesce them with sigmoid to achieve proper normalization.

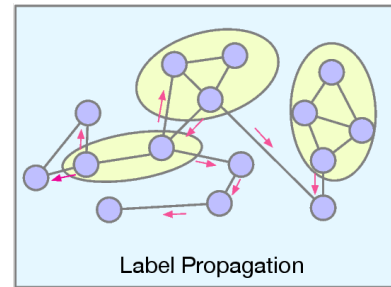


Figure 3. Label propagation: start with the circled nodes and propagate the labels following the arrows.

COMMUNITY SPLITTING

Instead of randomly selecting subgraphs, we also tried to split the graph into manageable subgraphs that are least connected. For detection of subgraphs, we looked into a variety of methods. A few promising ones are:

1. Girvan–Newman Algorithm (Girvan & Newman, 2002): detects communities by progressively removing edges based on the betweenness of the edges.
2. Louvain Method (Blondel et al., 2008): iteratively calculates the change in modularity that measures the density of links inside a community compared to links between communities.
3. Infomap (Rosvall & Bergstrom, 2008): Infomap optimizes the map equation, which accounts for flow patterns on the network (while modularity maximization approach does not).

See Appendix section B for the full list.

We first label each community separately, then merge community base on their probabilities, and biases \mathbf{b} . The idea is that for $\mathbf{b} = 0$, it is equally probable for the labels to flip i.e.:

$$\exp \mathbf{x}^T J \mathbf{x} = \exp ((-\mathbf{x})^T J (-\mathbf{x})) \quad (9)$$

since J is symmetric.

If $\|\mathbf{b}_{\text{subgraph}}\|_2 \leq \epsilon$, the probability for flipping the labels is high, then we determine whether to flip based on the aggregated edge between subgraphs starting with the subgraphs that have the highest $\|\mathbf{b}_{\text{subgraph}}\|_2$.

We compare the natural log probability of all combinations of subgraphs (x and $-x$) by computing the resulting MAP based on the new adjacency matrix J_{subgraph} , which is simply:

$$J_{\text{subgraph}} = S^T J S \quad (10)$$

where S is the community assignment matrix

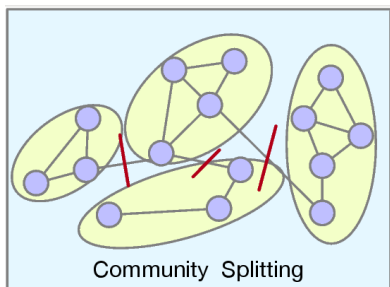


Figure 4. Community Splitting: Divide the community according to a community detection algorithm, then perform GNN on each one separately.

MAXIMUM SPANNING TREE

Another approximation direction is to approximate the whole graph to a tree that can minimize the information loss through these process. Thus we select the maximum spanning tree (MST). We find the maximum absolute weight spanning tree using Kruskal’s algorithm:

$$T = \arg \max_T \sum_{(i,j) \in T} |w_{i,j}| \quad (11)$$

The exact belief propagation can be applied on this MST.

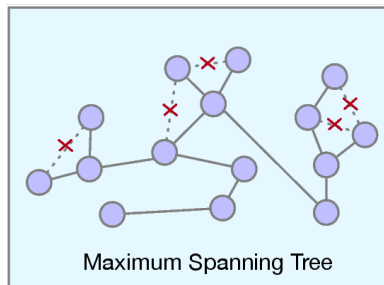


Figure 5. Maximum Spanning Tree: Find absolute weight spanning tree using Kruskal’s algorithm, then apply exact belief propagation.

EVALUATION OF METHODS

In order to see how well each of the methods performs (in combination with GNN) relative to approximate inference methods, we need a reliable way to estimate true labels, as at the scale of $n > 30$ nodes, exact inference is impossible to use. In order to do that, we will use belief propagation on trees, which is known to be exact in that case, and high-capacity MCMC (in particular, Gibbs sampling) with 10000 samples, 5000 burn-in and 100 stride to reduce auto-correlation.

4. Experimental results

In this section, we first describe experimental data collection and metrics, then experiments on the baseline model (Section 4.2), and then report the scalability experiments results (Section 4.3). We add some more experiments in the last subsection: these are less extensive, but could be useful next steps to take in the direction of making scalable GNN-based inference more accurate.

4.1. Experimental setup

Data collection. We obtain the data by generating a graph of a specific structure (such as a path or a fully-connected graph) and size n_{nodes} with weights on edges sampled independently from $N(0, 1)$; we use the resulting adjacency

matrix as the matrix $W \in \mathbb{R}^{n_{\text{nodes}} \times n_{\text{nodes}}}$ of pairwise interactions parameters, and additionally sample a vector $b \in \mathbb{R}^{n_{\text{nodes}}}$ of unary interactions from $N(0, (\frac{1}{4})^2)$. For each training task, we use a training set of 1300 graphs, and for testing (inference), we use 130 graphs.

For large graphs, as the exact inference is not tractable, we generate training data with MCMC algorithm. For each kind of graph structures, we generate graphs with 100 nodes.

Hardware. All presented experiments are run on a laptop.

Algorithm settings. All inference algorithms have been implemented by us and are available at https://github.com/ks-korovina/pgm_graph_inference.

For GNN architecture, we have written a sparse version of the default settings from (Yoon et al., 2018): the GNN’s hidden states (D) and messages (P) both have dimension 5, and both the message passing network \mathcal{M} and the readout network \mathcal{R} are two-layer MLPs with 64 units each. Training was run for 10 epochs (T) at learning rate 10^{-2} using Adam optimizer. While running BP inference, we use early stopping after 100 iterations or when the ℓ_2 distance between messages is less than 10^{-20} . For MCMC, the number of burn-in steps is set to 10000, while we provide data of sampling intervals 100 and 200 for comparison of the results.

4.2. Experiments with the baseline model

Since the code of the paper (Yoon et al., 2018) is not released, the first step is to write the code to reproduce part of its results. In the *in-sample* experiments, we see how GNN generalizes between graphs of the same size and structure; in *out-of-sample* experiments, we see how GNN generalizes to graphs of sizes and structures unseen in training, but still small enough for exact labelings. We also benchmark and interpret inference times and provide preliminary results for the MAP inference task.

In-sample experiment

We compare the methods’ performance on graphs of the same type and approximately the same size. We choose size 9 as in the paper (Yoon et al., 2018), and vary graph structures from more to less tree-like. For each test graph G and its node x_i , we plot its exact probability $p(x_i = +1)$ on the x-axis, and inferred probability $\hat{p}(x_i = +1)$ on the y-axis. The resulting scatter plots characterize comparative accuracy of different methods (the more concentrated around the diagonal the better). We note that BP is not exact even on tree graphs. We thoroughly checked our implementation and attribute this to accumulation of numerical error.

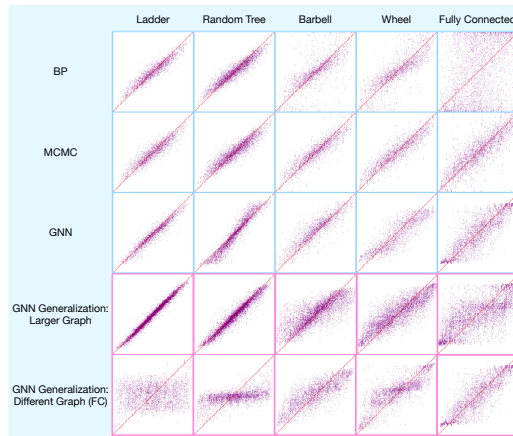


Figure 6. Generalization results for GNN trained on small MRFs

Out-of-sample experiment

For out-of-sample experiments, we focus on two kind of test samples. The first part is the generalization ability of GNN on large graphs of the same type. The second last line of Figure Figure 6 shows the results on ladder, random trees, barbell, wheel and fully-connected graphs. Comparing the figures, we observe that performance drastically drops for more dense graphs, but is still reasonable on trees. However, in all cases GNN still seems to perform better than BP.

For the second part, we assess GNN’s generalization ability to unseen structures in training. We use fully-connected graphs as the training samples, while testing on all kinds of graphs. The bottom line of Figure 6 shows the results. Comparing on these figures, we observe that the performance drops as the structure of testing graphs differs more from fully connected graphs. The generalization ability depends on the similarity between graph structures despite density of graphs.

Here, we compare the convergence of GGNN during training. See appendix A for other graph structures.

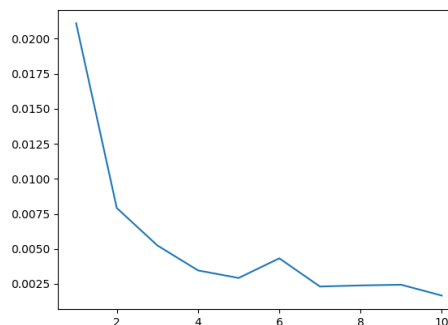


Figure 7. Training KLdiv (per epoch) on path graphs of size 9.

Inference Runtimes

Here, we compare inference times of the methods. Table 1 shows average inference time in seconds per graph. We see that as graph gets more dense, GNN becomes the fastest algorithm on inference. For sparser graphs, it loses on instantiation time (creating and loading the model). Table 2 for larger graphs shows similar results.

	sparse GGNN	BP	MCMC
Path	0.008	0.0037	0.1403
Star	0.0085	0.0015	0.133
Bipart	0.019	0.054	0.272
FC	0.0209	0.1301	0.2356

Table 1. Inference on graphs of size 9 (seconds per graph)

	sparse GGNN	BP	MCMC
Path/star	0.0098	0.0068	0.2797
Bipart/FC	0.0132	0.1537	0.3032

Table 2. Inference on graphs of size 15-17 (seconds per graph)

Training one epoch of Sparse GNN takes ≈ 20 seconds for 1300 training graphs, and between 1 and 10 minutes per epoch on 1500 graphs of size 100, depending on sparsity structure of the graphs, since the forward passing for Sparse GNN is linear in number of edges in the graph.

Map Inference

Figure 8 shows the results of MAP training: GNN exhibits comparable performance to other methods, while being orders of magnitude faster for dense graphs. As the graph structure becomes denser (containing more loops), all algorithms perform worse with GNN and MCMC still having higher MAP accuracy than BP (relative to sparse graph structure). Also as expected, GNN trained on structures closer fully connected is able to generalize better to fully connected. GNN is also able to generalize well to other and larger graph structures (not shown here).

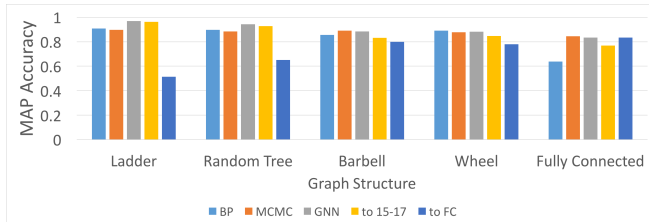


Figure 8. MAP Accuracy: Train on graph structure in the x-axis with 9 nodes and test on same graph structure with 9 nodes or 15-17 nodes as well as different graph structure. From left to right, the graph structure becomes denser, containing more loops.

4.3. Improving scalability

We have seen in section 4.2 that the baseline GNN is able to somewhat generalize from graphs of 9 to 16 nodes when the graph structure has been seen in training. We first see how far can we scale while still training on small graphs, and then detail on our plans for upcoming experiments.

Upscaling experiments on 25 nodes

In this experiment, we test how large can the inference graphs be for the baseline method. For testing this out-of-the-box scalability, we can still need to generate labeled “large” graphs for testing; this can be either done by exact enumeration (but only for up to ≈ 25 nodes, as it takes around 5 minutes per graph at this scale), or by only testing on trees, for which we can use BP.

Figure 9 shows a GNN model trained on path graphs of size 9. We generate and label only 10 testing graphs of size 25 due to time constraints. We see that in the simplest case of paths, the model is able to perform well. However, running this experiment on a fully-connected structure instead (Figure 10) shows that there is basically no generalization.

Figure 9. Generalization to path graphs of size 25

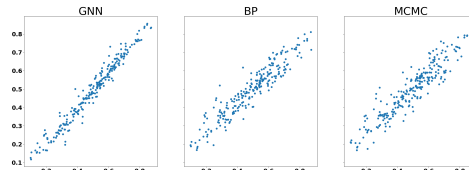
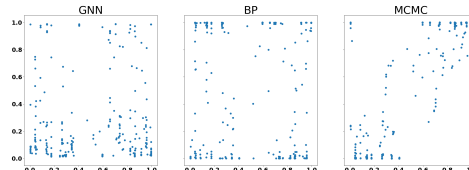


Figure 10. Generalization to FC graphs of size 25



Upscaling experiments on 100 nodes

We also conduct experiments the 100 nodes with semi-supervised training and co-training (see Section 3 for details). Results are shown in fig. 11.

Inference Time

The computing time of inference is also an important obstacle for large graph inference. We compare the per graph time on 15-17 nodes and 100 nodes on our implementation of sparse GGNN, belief propagation and MCMC. Results, shown in table 3 and table 4, demonstrate GNN is much faster than both BP and (1000-sample) MCMC, except for

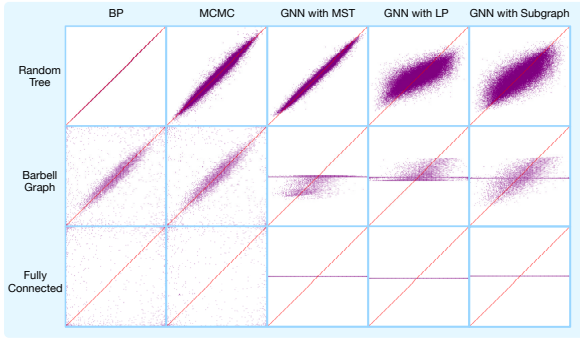


Figure 11. Generalization (in-sample) of GNN with approximate labeling methods vs other inference algorithms

the case of very sparse and small trees, where BP can be marginally better.

Table 3. Graphs of size 15-17 (sec/graph)

	sparse GGNN	BP	MCMC
Path/star	0.0098	0.0068	0.2797
Bipart/FC	0.0132	0.1537	0.3032

Table 4. Graphs of size 100 (sec/graph)

	sparse GGNN	BP	MCMC
Rand.tree	0.0142	0.0953	1.776
Barb+FC	0.1605	6.594	1.874

4.4. Additional experiments

DIFFERENT LOSS FUNCTION

In the previous experiments, we used reverse KL divergence as our loss function to train both marginals and MAP-inference. However, in the previous experiments, we observed an interesting artefact: for difficult graph structures, GNN predictions collapse to (0.5, 0.5). We realized that the reason for this might be implicit regularization of predictive distribution:

$$KL(p_{pred}, p_{true}) = \sum_{v=-1,1} p_{pred}(v) \log(p_{true}(v)) - \sum_{v=-1,1} p_{pred}(v) \log(p_{pred}(v))$$

minimizing which encourages higher-entropy p_{pred} . This regularization doesn't make much sense in our setting; so instead, we tried using regular cross-entropy (reverse of the first KL term above):

$$CE(p_{true}, p_{pred}) = \sum_{v=-1,1} p_{true}(v) \log(p_{pred}(v))$$

The results (in the case of GNN trained with subgraph labeling in Figure 12 and MST labeling in Figure 13) are given in Figure 12 and Figure 13. They show that the resulting predictions are still are weakly correlated to approximate truth for denser structures, but do not collapse to uniform distribution any more, and for barbells they improve noticeably.

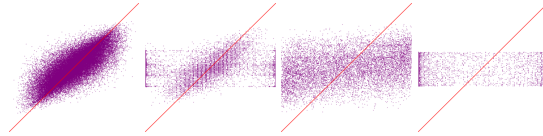


Figure 12. Subgraph labeling: random tree, barbell, wheel, fully-connected

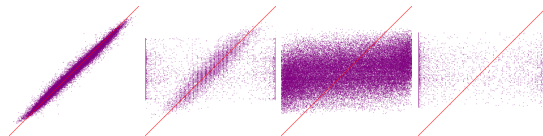


Figure 13. Max subtree labeling: random trees, barbell, wheel, fully-connected

CURRICULUM LEARNING

Curriculum training (Bengio et al., 2009) is the idea of designing a specific order of tasks for training the model, so that it is able to learn faster and generalize better. As all of labeling, training and inference on tree-like structures are faster, and generalizing on trees is an easier task, we could imagine first training the GNN on a larger set of trees labeled exactly (by BP), and then fine-tuning to a specific (more difficult) task of generalizing to denser structures. Unfortunately, on the dense structures that we tried, the results weren't anything interesting (we tried adding pre-training for 5 epochs on random trees, and it almost doesn't change the resulting plot), so we won't add them here and save properly exploring this idea for future work.

5. Conclusion

In this project, we explored a recently proposed idea for training graph neural networks to perform approximate inference on graphical models. We focused on the case of binary MRFs, implemented the gated GNN-based inference framework, as well as exact and approximate inference with BP (sparse and non-sparse versions), MCMC (Gibbs

sampling), all of which can be used for inference in both marginals and MAP inference modes on different graph sizes and structures. We conduct experiments on small graphs, on which comparing against exact inference is feasible, to evaluate GNN’s ability to generalize within and across graph sizes and structures. Then we attempt to resolve the largest drawback of the framework and evaluate three proposed approximate labeling methods. We see that for sparser structures, they are able to achieve some accuracy, and that they allow feasible training of GNN, which runs inference orders of magnitude faster than either BP or MCMC. We then briefly look at possible ways of improving accuracy of GNN, which includes changing the loss function and designing a curriculum procedure for GNN training. We conclude that although the work we have done is very preliminary, it can be a promising avenue for future work on fast and accurate approximate inference.

6. Ongoing work

6.1. Real-world Application on Image Segmentation

To demonstrate the usefulness of GNN as an inference machine, we plan to take image segmentation problem and using GNN to replace the inference procedure of CRF. Specifically, generating labels of fully connected CRF can be done efficiently via the algorithm in (Krähenbühl & Koltun, 2011). We train a GNN model using fully connected CRF and tree CRF with labels, and replace the CRF with trained GNN to do learning for image segmentation task. By using this trained GNN, we can use meta-learning technique to learn a better structure and dependency parameters of CRF. This can perhaps improve the task performance.

6.2. Information bottleneck

A potential evaluation metric is information bottleneck. Proposed by Tishby & Zaslavsky (2015), information bottleneck is the idea that mutual information between the layers and the input and output variables can give the optimal information theoretic limits for some networks. Recently, this technique has been adopted for interpretability by Professor Xing (Bang et al., 2019).

Acknowledgement

The team thanks their TA Maruan Al-Shedivat for helpful ideas, discussion and feedback.

References

- Bang, S., Xie, P., Wu, W., and Xing, E. Explaining a black-box using deep variational information bottleneck approach, 2019.
- Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.
- Bengio, Y., Louradour, J., Collobert, R., and Weston, J. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning, ICML ’09*, pp. 41–48, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-516-1. doi: 10.1145/1553374.1553380. URL <http://doi.acm.org/10.1145/1553374.1553380>.
- Blondel, V. D., Guillaume, J.-L., Lambiotte, R., and Lefebvre, E. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.
- Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., and Vandergheynst, P. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4): 18–42, 2017.
- Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212*, 2017.
- Girvan, M. and Newman, M. E. Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826, 2002.
- Hamilton, W., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pp. 1024–1034, 2017.
- Heess, N., Tarlow, D., and Winn, J. Learning to pass expectation propagation messages. In *Advances in Neural Information Processing Systems*, pp. 3219–3227, 2013.
- Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.
- Koller, D. and Friedman, N. *Probabilistic Graphical Models - Principles and Techniques*. MIT Press, 2009.
- Krähenbühl, P. and Koltun, V. Efficient inference in fully connected crfs with gaussian edge potentials. In *Advances in neural information processing systems*, pp. 109–117, 2011.

Lin, G., Shen, C., Reid, I., and van den Hengel, A. Deeply learning the messages in message passing inference. In *Advances in Neural Information Processing Systems*, pp. 361–369, 2015.

Newman, M. E. J. Finding community structure in networks using the eigenvectors of matrices. 2006. doi: 10.1103/PhysRevE.74.036104.

Pons, P. and Latapy, M. Computing communities in large networks using random walks (long version), 2005.

Ramakrishna, V., Munoz, D., Hebert, M., Bagnell, J. A., and Sheikh, Y. Pose machines: Articulated pose estimation via inference machines. In *European Conference on Computer Vision*, pp. 33–47. Springer, 2014.

Reichardt, J. and Bornholdt, S. Statistical mechanics of community detection. 2006. doi: 10.1103/PhysRevE.74.016110.

Ross, S., Munoz, D., Hebert, M., and Bagnell, J. A. Learning message-passing inference machines for structured prediction. In *CVPR 2011*, pp. 2737–2744, June 2011. doi: 10.1109/CVPR.2011.5995724.

Rosvall, M. and Bergstrom, C. T. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences*, 105(4): 1118–1123, 2008. ISSN 0027-8424. doi: 10.1073/pnas.0706851105. URL <https://www.pnas.org/content/105/4/1118>.

Schlichtkrull, M., Kipf, T. N., Bloem, P., van den Berg, R., Titov, I., and Welling, M. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, pp. 593–607. Springer, 2018.

Tishby, N. and Zaslavsky, N. Deep learning and the information bottleneck principle, 2015.

Yoon, K., Liao, R., Xiong, Y., Zhang, L., Fetaya, E., Urtasun, R., Zemel, R., and Pitkow, X. Inference in probabilistic graphical models by graph neural networks, 2018.

Zheng, S., Jayasumana, S., Romera-Paredes, B., Vineet, V., Su, Z., Du, D., Huang, C., and Torr, P. H. Conditional random fields as recurrent neural networks. In *Proceedings of the IEEE international conference on computer vision*, pp. 1529–1537, 2015.

A. Convergence Plots

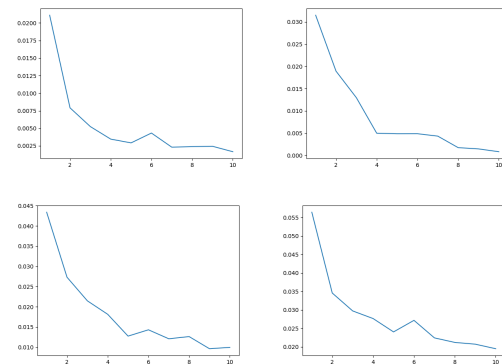


Figure 14. Training KLdiv (per epoch) on {path,star,bipart,fc} graphs of size 9

B. Community Detection methods

Table 5. List of community detection algorithms

Algorithm	Description
Optimal Modularity	Using linear programming, it is possible to solve a large integer optimization problem to find the optimal community structure. However, it is very slow, especially for large graphs.
Girvan–Newman (Girvan & Newman, 2002)	Detects communities by progressively removing edges based on the edge betweenness.
Walktrap (Pons & Latapy, 2005)	Based on random walks, it is likely to end up in the same community after taking some random steps.
Leading Eigenvector (Newman, 2006)	Newman’s leading eigenvector method (from the graph Laplacian) that splits the community structure recursively by maximizing the modularity of the starting network.
Spinglass (Reichardt & Bornholdt, 2006)	Finds the community structure by interpreting it as as finding the ground state of an infinite range spin glass in condensed matter physics. In practice, this requires a lot of tuning.
Louvain Method (Blondel et al., 2008)	Bottom-up algorithm that starts with every vertex belonging to a separate community, and merges them to maximize the vertices local contribution to the overall modularity score.
Infomap (Rosvall & Bergstrom, 2008)	optimizes the map equation, which accounts for flow patterns on the network (while modularity maximization approach does not). Hence, it is well-suited for bibliometric networks.