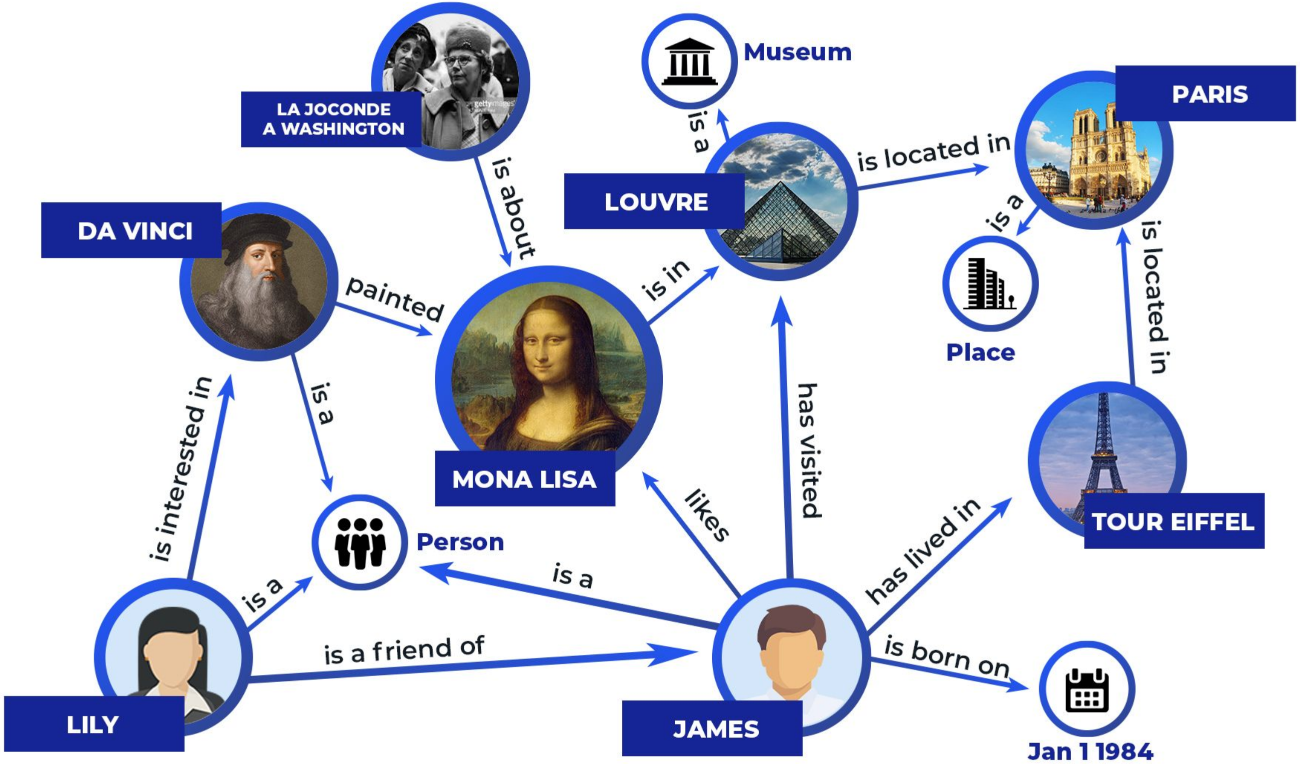
Efficient Probabilistic Logic Reasoning with Graph Neural Networks

Boyuan He
Haochen Li
Jingyue Shen

Schedule

- Background - knowledge graph & MLN
- Motivation
- Model
- Experiment

Knowledge Graph



Knowledge Graph

Knowledge Graph is a tuple $\mathcal{K} = (\mathcal{C}, \mathcal{R}, \mathcal{O})$

$\mathcal{C} = \{c_1, \ldots, c_M\}$ -- set of entities/constants

$\mathcal{R} = \{r_1, \ldots, r_N\}$ -- set of relations/predicates

$\mathcal{O} = \{o_1, \ldots, o_L\}$ -- set of observed facts

Knowledge Graph

Predicate is a logic function $\mathcal{C} \times \ldots \times \mathcal{C} \mapsto \{0,1\}$

$r(c, c')$ -- c have relation with c' (asymmetric)

Ground predicate is predicate with a set of entities assigned

$a_r = (c, c')$ -- assignment ($r(c, c') \rightarrow r(a_r)$)

ground predicate ≡ binary random variable

Observed fact is truth value {0, 1} assigned to a ground predicate

$\mathrm{L}(c, c') = 1$

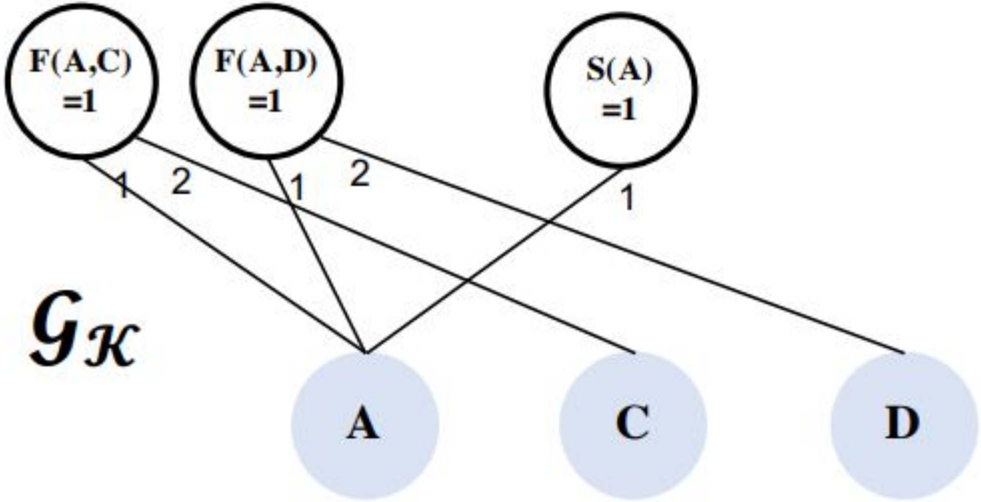
Knowledge Graph

knowledge base $\mathcal{K}$ represented by a bipartite graph $\mathcal{G}_{\mathcal{K}} = (\mathcal{C}, \mathcal{O}, \mathcal{E})$

$\mathcal{C}$ -- constant

$\mathcal{O}$ -- observed facts (factor)

$\mathcal{E}$ -- a set of edge



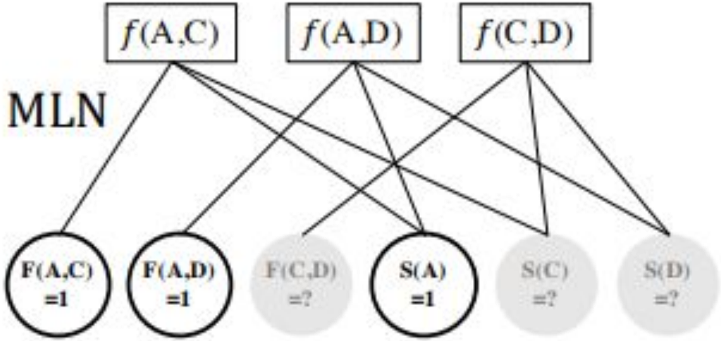
Markov Logic Networks

MLNs use logic formulae to define potential functions in undirected graphical models

logic formulae -- $f(\cdot) : \mathcal{C} \times \ldots \times \mathcal{C} \mapsto \{0, 1\}$ defined by composition of predicates

Example

$$f(c, c') := \texttt{Smoke}(c) \wedge \texttt{Friend}(c, c') \Rightarrow \texttt{Smoke}(c')$$



Markov Logic Networks

MLN can be represented as a joint distribution over all observed facts O and unobserved facts H

$$P_w(\mathcal{O}, \mathcal{H}) := \frac{1}{Z(w)} \exp\left(\sum_{f \in \mathcal{F}} w_f \sum_{a_f \in \mathcal{A}_f} \phi_f(a_f)\right)$$

$\mathcal{O}$ -- observed facts, $\mathcal{H}$ -- unobserved facts

$a_r = (c, c')$ -- assignment (similar to KG)

$\mathcal{A}_f = \{a_f^1, a_f^2, \ldots\}$ -- entire collection of consistent assignments

$\phi_f(\cdot)$ potential function defined by a formula f

w_f -- confidence score of formula f

$Z(w)$ -- partition function summing over all ground predicates

KG vs MLN

Knowledge graphs	MLN
Sparse Number of edges linear to the number of entities	Denser number of edges high-order polynomials to the number of entities number of nodes can be quadratic or more to the number of entities

Motivation

Markov Logic Network

Pros:

1) Incorporate logic rules as prior knowledge

2) Allow MLN to generalize in tasks with small amount of labeled data

Cons

Inference in MLN is computationally intensive

Motivation

GNN-based methods

Pros:

1) Can achieve good performance with sufficient labeled data

Cons

1) Long-tail nature of knowledge graph leads to data scarcity problem among long-tail relations

- long-tail property: a large portion of the relations in only a few triplets.

Model

- A method which is data-driven yet can still exploit the prior knowledge encoded in logic rules.

- Use a self-designed GNN as the inference module to make MLN’s inference efficient

Model

MLN - trained with variational EM framework

E-step (inference)

M-step (learning)

Use GNN to approximate the posterior distribution $P(H|O)$

Optimization objective for MLN

- Instead of optimizing the log-likelihood of all the observed facts $\log P_w(\mathcal{O})$

- Optimize the variational evidence lower bound (ELBO) of the data log-likelihood:

$$\log P_w(\mathcal{O}) \geqslant \mathcal{L}_{\mathrm{ELBO}}(Q_\theta, P_w) := \mathbb{E}_{Q_\theta(\mathcal{H}|\mathcal{O})}\Big[\log P_w(\mathcal{O}, \mathcal{H})\Big] - \mathbb{E}_{Q_\theta(\mathcal{H}|\mathcal{O})}\Big[\log Q_\theta(\mathcal{H}|\mathcal{O})\Big]$$

$Q_\theta(\mathcal{H} \mid \mathcal{O})$ -- variational posterior distribution of the latent variables

$P_w(\mathcal{O}, \mathcal{H})$ -- joint distribution modeled by MLN

Derivation of lower bound

- f(E[X]) >= E[f(X)] (Jensens Inequality)
- $$\begin{aligned} \log p(x) &= \log \int_z p(x,z) \\ &= \log \int_z p(x,z) \frac{q(z)}{q(z)} \\ &= \log \left(\mathrm{E}_q \left[\frac{p(x,Z)}{q(z)} \right] \right) \\ &\geq \mathrm{E}_q[\log p(x,Z)] - \mathrm{E}_q[\log q(Z)]. \end{aligned}$$

Variational EM framework for MLN

$$\mathcal{L}_{\text{ELBO}}(Q_\theta, P_w) := \mathbb{E}_{Q_\theta(\mathcal{H}|\mathcal{O})}\Big[\log P_w\left(\mathcal{O},\mathcal{H}\right)\Big] - \mathbb{E}_{Q_\theta(\mathcal{H}|\mathcal{O})}\Big[\log Q_\theta\left(\mathcal{H}|\mathcal{O}\right)\Big]$$

M-step: learn the weights of the logic formulae in MLN, where Q_θ is fixed and P_w is optimized to maximize the data log-likelihood

E-step: Infer the posterior distribution of the latent variables, where P_w is fixed and Q_θ is optimized to minimize the KL divergence between Q_θ (H|O) and P_w (H|O)

Variational EM algorithm E-step

- Use mean-field variational distribution to approximate the posterior distribution, each unobserved ground predicate inferred as follows:

$$Q_\theta(\mathcal{H}|\mathcal{O}) := \prod_{r(a_r)\in\mathcal{H}} Q_\theta(r(a_r))$$

$Q_\theta(r(a_r))$ -- factorized distribution (follow bernoulli distribution)

- Q_θ is parameterized using ExpressGNN

More about variational inference:
https://www.cs.cmu.edu/~epxing/Class/10708-17/notes-17/10708-scribe-lecture13.pdf

Variational EM algorithm E-step

After parameterization, $\mathcal{L}_{\text{ELBO}}(Q_\theta, P_w)$ can be rewritten as:

$$\left(\sum_{f \in \mathcal{F}} w_f \sum_{a_f \in \mathcal{A}_f} \mathbb{E}_{Q_\theta(\mathcal{H}|\mathcal{O})} \Big[\phi_f(a_f) \Big] - \log Z(w) \right) - \left(\sum_{r(a_r) \in \mathcal{H}} \mathbb{E}_{Q_\theta(r(a_r))} \Big[\log Q_\theta(r(a_r)) \Big] \right)$$

Variational EM algorithm E-step

$$\left(\boxed{\sum_{f \in \mathcal{F}} w_f \sum_{a_f \in \mathcal{A}_f}} \mathbb{E}_{Q_\theta(\mathcal{H}|\mathcal{O})}\left[\phi_f(a_f)\right] - \log Z(w)\right) - \left(\boxed{\sum_{r(a_r) \in \mathcal{H}}} \mathbb{E}_{Q_\theta(r(a_r))}\left[\log Q_\theta(r(a_r))\right]\right)$$

Sum over all formula and all possible assignment

Sum over all assignments to latent variables

Problem: intractable computational cost

Solution:

- The author uses mini-batches of ground formulae
- In each optimization iteration
  - Sample a batch of ground formulae
  - For each formulae sampled, compute the objective w.r.t involved latent variables

Variational EM algorithm E step

- If the task have sufficient label data, add supervised learning objective to enhance the inference network

$$\mathcal{L}_{\text{label}}(Q_\theta) = \sum_{r(a_r)\in\mathcal{O}} \log Q_\theta(r(a_r))$$

- It is complementary to ELBO on predicates that are not well covered by logic rules but have enough observed facts

- And the overall objective function is

$$\mathcal{L}_\theta = \mathcal{L}_{\text{ELBO}}(Q_\theta, P_w) + \lambda\mathcal{L}_{\text{label}}(Q_\theta)$$

Variational EM algorithm M step

$$\mathcal{L}_{\text{ELBO}}(Q_\theta, P_w) := \mathbb{E}_{Q_\theta(\mathcal{H}|\mathcal{O})}\Big[\log P_w(\mathcal{O}, \mathcal{H})\Big] - \mathbb{E}_{Q_\theta(\mathcal{H}|\mathcal{O})}\Big[\log Q_\theta(\mathcal{H}|\mathcal{O})\Big]$$

- learn the weights of logic formulae in MLN with $Q_\theta(\mathcal{H}|\mathcal{O})$ fixed
- $Z(w)$ is no longer a constant, and has exponential number of terms, so is intractable to directly optimize t $P_w(\mathcal{O}, \mathcal{H})$
- Use pseudo-log-likelihood: $\log P_w(O) \approx \sum_i \log P_w(o_i | o_{N(i)})$
- The optimization goal becomes:

$$P_w^*(\mathcal{O}, \mathcal{H}) := \mathbb{E}_{Q_\theta(\mathcal{H}|\mathcal{O})}\Big[\sum_{r(a_r)\in\mathcal{H}} \log P_w(r(a_r) \mid \text{MB}_{r(a_r)})\Big]$$

$\text{MB}_{r(a_r)}$ -- Makarov blanket of $r(a_r)$

Variational EM algorithm M step

- For each formula i that connects $r(a_r)$ to its Markov blanket, optimize weight w_i
- Use gradient descent, with the derivative:

$$\nabla_{w_i} \mathbb{E}_{Q_\theta}[\log P_w(r(a_r) \mid \mathrm{MB}_{r(a_r)})] \simeq y_{r(a_r)} - P_w(r(a_r) \mid \mathrm{MB}_{r(a_r)})$$

$y_{r(a_r)}$ = 0 or 1, if $r(a_r)$ is observed fact

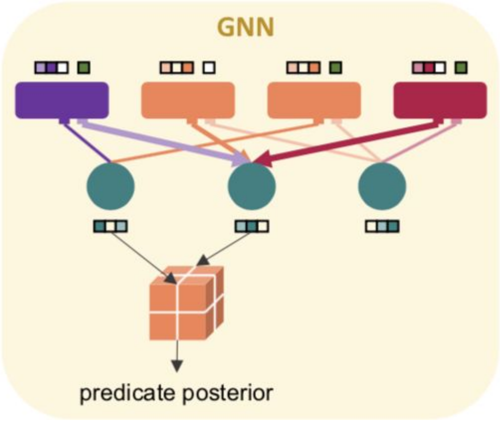
= $Q_\theta(r(a_r))$, otherwise

ExpressGNN

Need to have a expressive and efficient network to approximate true posterior distribution in step E

ExpressGNN involve three parts:

1. Vanilla graph neural network (GNN)
2. tunable embeddings
3. uses the embeddings to define the variational posterior



Step 1: Vanilla GNN

- Build a GNN on the knowledge graph $\mathcal{G}_{\mathcal{K}}$, which is much smaller than MLN.
- Parameters $\boldsymbol{\theta}_1$ $\boldsymbol{\theta}_2$ are shared across the entire graph and independent of # of entities
- The GNN is a compact model with $O(d^2)$ parameters given d dimensional embeddings

Algorithm 1: GNN()

Initialize entity node: $\mu_c^{(0)} = \mu_0, \ \forall c \in \mathcal{C}$

for $t = 0$ *to* $T - 1$ **do**

  ▷ Compute message $\forall r(c, c') \in \mathcal{O}$

  $m_{c' \to c}^{(t)} = \mathtt{MLP}_1(\mu_{c'}^{(t)}, r; \boldsymbol{\theta}_1)$

  ▷ Aggregate message $\forall c \in \mathcal{C}$

  $m_c^{(t+1)} = \mathtt{AGG}(\{m_{c' \to c}^{(t)}\}_{c' : r(c,c') \in \mathcal{O}})$

  ▷ Update embedding $\forall c \in \mathcal{C}$

  $\mu_c^{(t+1)} = \mathtt{MLP}_2(\mu_c^{(t)}, m_c^{(t+1)}; \boldsymbol{\theta}_2)$

return embeddings $\{\mu_c^{(T)}\}$

Step 2: Add tunable weights

- For each entity in the KG, augment its GNN embedding with a tunable embedding $\boldsymbol{\omega}_c \in \mathbb{R}^k$ as $\hat{\mu}_c = [\mu_c, \boldsymbol{\omega}_c]$
- The tunable embeddings increase the expressiveness of the model
- Number of parameters in tunable embeddings is $O(kM)$

Step 3: Calculate posterior

- variational posterior defined as:

$$Q_\theta(r(c_1, c_2)) = \sigma(\mathtt{MLP}_3(\hat{\mu}_{c_1}, \hat{\mu}_{c_2}, r; \boldsymbol{\theta}_3))$$

- number of parameters in $\boldsymbol{\theta}_3$ is $O(d+k)$

ExpressGNN

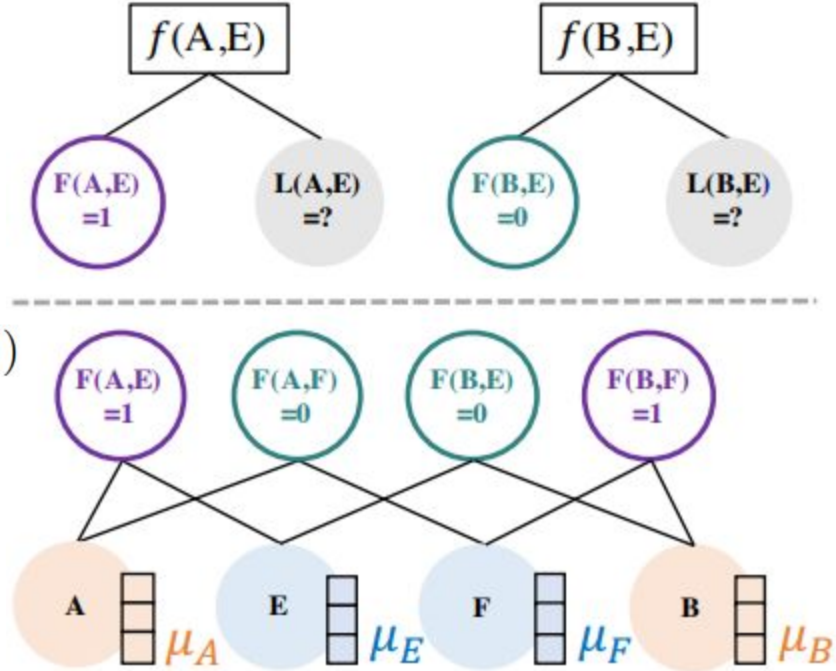
- Compact GNN assigns similar embeddings to similar entities in the KG
- Expressive tunable embeddings allows encoding of entity specific information beyond graph structures
- Overall number of trainable parameters is $O(d^2 + kM)$, and by controlling d and k, we can control the trade-off between compactness and expressiveness

Why we need the tunable embedding

Vanilla GNN produces the same embedding for some nodes that should be distinguished

for example given two predicates (Friend: F(·, ·), Like: L(·, ·)), and one formula (F(c, c′) ⇒ L(c, c′)):

- A and B have opposite relations with E, but GNN will produce same embedding
- $\mathtt{L}(A, E)$and$\mathtt{L}(B, E)$ have different posteriors, but they get same $Q_\theta(\mathtt{L}(A, E))$
- ExpressGNN avoid this problem by allowing additional tunable embeddings



ExpressGNN

Efficient: works on the knowledge graph, instead of the huge MLN grounding graph, more efficient than the existing MLN inference methods

Compact: the GNN model with shared parameters can be very memory efficient

Expressive: the GNN model can capture structure knowledge in the knowledge graph, and the tunable embeddings can encode entity-specific information

Generalizable: with the GNN embeddings, ExpressGNN may generalize to new entities or even different but related knowledge graphs

Experimental Setup

Benchmark datasets

UW-CSE , Cora, synthetic Kinship datasets, and FB15K-237

General settings

Linux machine with RTX 2080 Ti, Intel Xeon Silver 4116 and 256GB RAM

Hyperparameters:

use 0.0005 as the initial learning rate, and decay it by half for every 10 epochs without improvement

two-layer MLP with ReLU activation function for each embedding update step

MLP parameters is different for steps, edge types and direction of embedding aggregation

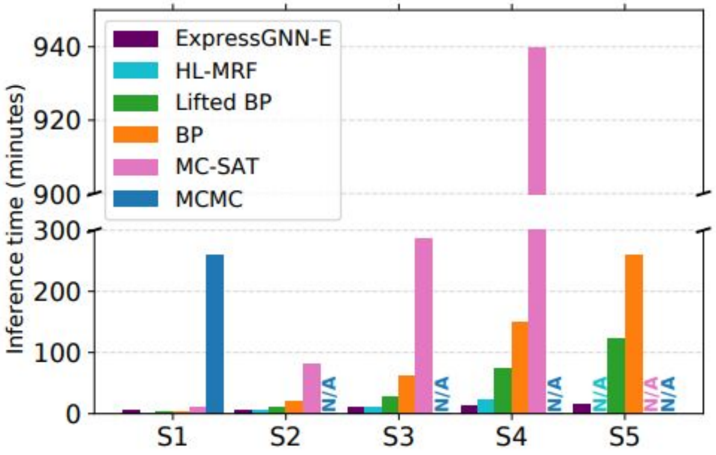
...

Experiment Result

Method	Kinship					UW-CSE					Cora
	S1	S2	S3	S4	S5	AI	Graphics	Language	Systems	Theory	(avg)
MCMC	0.53	-	-	-	-	-	-	-	-	-	-
BP / Lifted BP	0.53	0.58	0.55	0.55	0.56	0.01	0.01	0.01	0.01	0.01	-
MC-SAT	0.54	0.60	0.55	0.55	-	0.03	0.05	0.06	0.02	0.02	-
HL-MRF	**1.00**	**1.00**	**1.00**	**1.00**	-	0.06	0.06	0.02	0.04	0.03	-
ExpressGNN-E	0.97	0.97	0.99	0.99	0.99	**0.09**	**0.19**	**0.14**	**0.06**	**0.09**	**0.64**

Experiment Result

Method	Inference Time (minutes)				
	AI	Graphics	Language	Systems	Theory
MCMC	>24h	>24h	>24h	>24h	>24h
BP	408	352	37	457	190
Lifted BP	321	270	32	525	243
MC-SAT	172	147	14	196	86
HL-MRF	135	132	18	178	72
ExpressGNN-E	**14**	**20**	**5**	**7**	**13**



Experiment Result

Configuration	Cora				
	S1	S2	S3	S4	S5
Tune64	0.57	0.74	0.34	0.55	0.70
GNN64	0.57	0.58	0.38	0.54	0.53
GNN64+Tune4	0.61	0.75	0.39	0.54	0.70
Tune128	**0.62**	0.76	0.42	**0.60**	0.73
GNN128	0.60	0.59	0.45	0.55	0.61
GNN64+Tune64	**0.62**	**0.79**	**0.46**	0.57	**0.75**

Experiment Result

Model	MRR					Hits@10				
	0%	5%	10%	20%	100%	0%	5%	10%	20%	100%
MLN	-	-	-	-	0.10	-	-	-	-	16.0
NTN	0.09	0.10	0.10	0.11	0.13	17.9	19.3	19.1	19.6	23.9
Neural LP	0.01	0.13	0.15	0.16	0.24	1.5	23.2	24.7	26.4	36.2
DistMult	0.23	0.24	0.24	0.24	0.31	40.0	40.4	40.7	41.4	48.5
ComplEx	0.24	0.24	0.24	0.25	0.32	41.1	41.3	41.9	42.5	51.1
TransE	0.24	0.25	0.25	0.25	0.33	42.7	43.1	43.4	43.9	52.7
RotatE	0.25	0.25	0.25	0.26	0.34	42.6	43.0	43.5	44.1	53.1
pLogicNet	-	-	-	-	0.33	-	-	-	-	52.8
ExpressGNN-E	**0.42**	**0.42**	0.42	0.44	0.45	53.1	53.1	53.3	55.2	57.3
ExpressGNN-EM	**0.42**	**0.42**	**0.43**	**0.45**	**0.49**	**53.8**	**54.6**	**55.3**	**55.6**	**60.8**

Demo

Quiz

1. What is a potential problem for applying GNN-based methods on knowledge graphs? (single choice)

a. There is no potential problem. GNN can perform well on all graph-related problems.
b. Long-tail nature of Knowledge Graph makes applying GNN hard since GNN requires sufficient labeled instances to achieve good performance.
c. Unobserved features in Knowledge Graph could never be learned by any neural network through any means.

2. Which of the following is true about Markov Logic Network(MLN)? (single choice)

a. Inference in MLN is computationally intense.
b. The MLN structure is normally sparser than knowledge graph
c. a and b

3. In regards to ExpressGNN, at which step is the GNN used in variational EM for Markov Logic Network(MLN)? (single choice)
a. E step
b. M step

Questions