CuriousWalk: Enhancing Multi-Hop Reasoning in Graphs with Random Network Distillation

Varun Kausika
CFS AI Research, Dell Inc., USA
varun.kausika@dell.com

Saurabh Jha
CFS AI Research, Dell Inc., USA

Adya Jha
CFS AI Research, Dell Inc., USA

Amy Zhang
University of Texas at Austin

Michael Sury
McCombs School of Business

Abstract

Structured knowledge bases in the forms of graphs often suffer from incompleteness and inaccuracy in representing information. One popular method of densifying graphs involves constructing a reinforcement learning agent that learns to traverse entities and relations in a sequential way from a query entity, according to a query relation until it reaches the desired answer entity. However, these agents are often limited by sparse reward structures of the environment, as well as their inability to find diverse paths from the question to the answer entities. In this paper, we attempt to address these issues by augmenting the agent with intrinsic rewards which can help in exploration as well as offering meaningful feedback at intermediate steps to push the agent in the right direction.

1 Introduction

The landscape of multi-hop reasoning in knowledge graphs has seen continued innovation with the introduction of novel reinforcement learning methods. This refers to the ability of an intelligent system to make inferences by traversing multiple "hops" through interconnected pieces of information, such like those present in knowledge graphs or relational datasets where entities are linked by various relationships.

Existing attempts at multi-hop reasoning fall broadly two categories: those that predict relations given query entities and answer entities, and those that predict answer entities given query entities and relations. Relation prediction methods [Xiong et al., 2018, Chen et al., 2018, Neelakantan et al., 2015, Das et al., 2017, Lao et al., 2011, Wang et al., 2019] are common in tasks such as link prediction whereas entity prediction methods [Shen et al., 2018, Lin et al., 2018, Lv et al., 2019, Das et al., 2018, Li and Cheng, 2019, Wan and Du, 2021] are useful in query answering tasks.

Of the entity prediction methods, several are based on on the reinforcement learning paradigm. In Das et al. [2018], path finding capabilities of the agent have been prioritized. Specifically, the REINFORCE algorithm has been used to train an end-to-end question answering agent. In other work, hierarchical reinforcement learning has been proposed [Wan et al., 2020] to decompose relations in sub-tasks in order for the agent to understand actions better in different contexts. This helps in solving the multiple semantic issue, where relations may occur many times in the relation vocabulary, but may mean different things depending on it's surroundings. GaussianPath [Wan and Du, 2021] explores a bayesian algorithm where the uncertainty of different hops are expressed using a posterior,

NeurIPS 2023 New Frontiers in Graph Learning Workshop (NeurIPS GLFrontiers 2023).

which is derived from a prior and likelihood which is provided by an oracle. Thompson sampling is conducted over the posterior distributions to incorporate exploration and diversity. However, the reward structure of these environments offer delayed binary rewards, and may not be generalizable.

Current literature does attempt to address sparsity. For example, Dual Agents [Zhang et al., 2021] have been applied to interactively perform coarse and fine reasoning. Here, a mutual reinforcement reward has been constructed to help the coarse reasoner provide hints to the fine reasoner, as well as learn to traverse trajectories consistent with each other. Adversarial multi-agent systems have also emerged as a promising direction for future research. In particular, agents have been designed to extract arguments from graphs and debate over them [Hildebrandt et al., 2020], with a common judge to classify the winner as well as process the arguments and provide rewards in the form of score representations which take the arguments as inputs. In reward shaping [Lin et al., 2018], sparsity is mitigated with the assumption that all intermediate rewards take the form of a fact function which is known beforehand [Trouillon et al., 2016, Dettmers et al., 2018].

In our work, we attempt to generate intrinsic rewards using random network distillation [Burda et al., 2018], which is a curiosity based method originally intended to improve intrinsic curiosity module [Pathak et al., 2017] based methods in highly stochastic environments. To the best of our knowledge, CuriousWalk is the first method that uses an intrinsic reward to improve reasoning performance.

2 Method

We present a novel approach to enhancing multi-hop reasoning in knowledge graphs through Random Network Distillation. Our method addresses the challenges of sparsity mitigation and improves the diversity of identified paths, which have been limitations in existing techniques.

2.1 Environment

Our goal is to be able to answer queries of the form $(e_q, r, ?)$. We formulate the environment as a partially observable Markov decision process. Following MINERVA [Das et al., 2018], the problem can be represented using a knowledge graph derived from a knowledge base. Within this graph, we define states, observations, actions, transitions and extrinsic rewards as follows:

1. **States** are described by the tuple (e_q, r_q, e_t, e_a), where the query entity, query relation and answer entity are e_q, r_q and e_a respectively. In addition to these, the current location of the agent e_t is also included in the state.
2. **Observation** for a given state S is given by (e_q, r_q, e_t).
3. **Actions** at a exploratory entity e_t is defined by all the outgoing relations from e_t. If the current location is sufficient, a self-looping action can be taken. Furthermore, inverse relations for undoing wrong actions is also possible.
4. **Transitions** in our POMDP transitions are deterministic and consist of updating the exploratory state e_t according to the agent's action.
5. **Extrinsic rewards** obtained at a given exploratory state $e_t \in S$ depend on the true labels of answer entities e_a and are formulated as:

$$R(S) = \begin{cases} 1 & e_t = e_a \\ 0 & e_t \neq e_a \end{cases}$$

2.2 Intrinsic rewards

In order to enhance the reward structure, we employ random network distillation [Burda et al., 2018]. We prefer this method over intrinsic curiosity module (ICM) methods because ICM uses state information present in consecutive time steps to generate rewards, and would not work well when the environment is unpredictable, due to the popularly known noisy-TV problem. An overall reward which combines both intrinsic and extrinsic rewards is calculated later.

2.2.1 Target network

The target network is a neural network that serves as a reference or teacher network, It is used to estimated the expected intrinsic reward the agent should receive for it's current state-action pairs. The target network weights are initialized randomly and are not trained. Our network architecture can formally be written as:

$$f(S) = W_3 ReLU(W_2 ReLU(W_1[e_t; a_t]))$$

Where f is the target network feature representation and ; denotes vector concatenation. Additionally , the network takes the actions possible at the exploratory state S, i.e. the outgoing relations as inputs. The hidden layers all have 128 units and the output has 64 units.

2.2.2 Predictor network

This network estimates the intrinsic reward and drives the agent's exploration. The weights are initialized randomly and are trained according to a loss function who's target is defined by the target network. The predictor network architecture can formally be written as:

$$\hat{f}(S) = W_2 ReLU(W_1[e_t; a_t])$$

where f is the predictor network feature representation, and has one hidden layer with 64 hidden units, and 64 output units.

2.2.3 Training the predictor network

The predictor network tries to match the target network's output for a given state S. We found that the L2 loss function between the target and predictor network works well, as given by:

$$L(f(S), \hat{f}(S)) = ||f(S) - \hat{f}(S)||_2^2$$

This is also the intrinsic reward at state S for our agent. As training goes on, the predictor network makes small errors on familiar states and larger errors on unfamiliar states. This translates into a higher reward for unexplored areas of the graph.

2.3 History, relation and entity embeddings

Since the knowledge graph is incomplete, embeddings [Trouillon et al., 2016, Dettmers et al., 2018] are used to mitigate the issue of rewarding false negatives the same as true negatives [Lin et al., 2018]. The history of states and actions taken by the agent in an episode as input to an LSTM along with the observation and action at $t-1$ to forecast the encoded history at t. This encoded history is used in the policy network later [Das et al., 2018].

$$h_t = LSTM(h_{t-1}, [a_{t-1}; o_t])$$

The actions taken by the agent are embedded into a action embedding space, which comprises of the outgoing relation embedding and the destination entity embedding. Both relation and entity embedding spaces have embedding vector length d.

2.4 Policy network

The policy network was trained to maximize the cumulative expected return as:

$$\theta = max_\theta \, \mathbb{E}_{(e_1,r,e_2)\sim D} \, \mathbb{E}_{A_1,\ldots,A_{T-1}\sim\pi_\theta}[G_1|S_1 = (e_q, e_q, r, e_a)]$$

Where θ are the parameters of policy π. An underlying distribution D is assumed for transitions (e_1, r, e_2). G_t, the goal is defined as:

$$G_t = R_{c,t} + \gamma R_{c,t+1} + \gamma^2 R_{c,t+2} + ... + \gamma^{T-t+1} R_{c,T}$$

Where R_c is the combined reward and γ is the discount factor. The first expectation is replaced with sample average approximation and for the second expectation several trajectories are rolled out for each state [Das et al., 2018]. The policy network is stochastic over the actions available and uses a feedforward neural network as below.:

$$d_t = softmax(A_t(W_2 ReLU(W1[h_t; o_t; r_q])))$$
$$A_t \sim Categorical(d_t)$$

Where the query relation r_q gives context to the network in selecting actions. Overall, the trainable weights of our policy include both the LSTM weights as well as the policy network weights.

3 Training the agent

The agent was trained using a modified version of REINFORCE [Williams, 1992], as shown below. The goal of training is to maximize the expected cumulative reward over trajectories that were generated from the query entity until the terminal length of the path. Training can be divided into two stages, namely the pretraining of the predictor network, and the simultaneous training of both the predictor network and the policy network. Note that it is important to continue training of the predictor network in the second phase, as repeating observations should receive lower intrinsic rewards over time.

Algorithm 1 REINFORCE with RND pseudocode

Sample input $I = (e_t, r_{t+1}, e_{t+1})$ from validation set
for all $I_i \in I$ **do**
  Get $\hat{f}(I)$ and $f(I)$ from predictor and target networks
  Calculate loss $L(f(S_t), \hat{f}(S_t)) = ||f(I_i) - \hat{f}(I_i)||_2^2$
  Train predictor weights θ_p w.r.t backpropagated loss
end for
for $Episode$ in $Episodes$ **do**
  for t in $(1, 2, ..., T)$ **do** ▷ T = number of time steps in trajectory
    Sample action from policy $A_t \sim \pi(a|S_t; \theta_\pi)$
    Get new state and extrinsic reward $S_{t+1}, R_{E,t+1} \sim p(S_{t+1}, R_{E,t+1}|S_t, A_t)$
    Calculate intrinsic reward $R_{I,t} = ||f(I_i) - \hat{f}(I_i)||_2^2$
    Train predictor network on current (e_t, r_{t+1}, e_{t+1})
    Normalize all intrinsic rewards: $R_{I,t} \leftarrow (R_{I,t} - \mu_{r_I})/\sigma_{r_I}$
    Calculate expected return $G_t = \Sigma_{k=t}^{T} \gamma^{k-t+1}(r_{E,k} + r_{I,k})$
    Update policy weights θ_π: $\theta_\pi \leftarrow \theta_\pi + \alpha G_t \gamma^t \nabla_{\theta_\pi} log(\pi(A_t|S_t; \theta_\pi))$
  end for
end for

3.1 Pretraining predictor network

The predictor network was pretrained over samples from the validation set, since exposing the predictor network to training samples directly might lead to overfitting. Although the number of samples in the validation set is low, pretraining the network helps it "get used to" matching the target network's representation, and familiarize itself with different states. This led to an appreciable improvement in the first few hops our agent takes in the second stage, since MINERVA is vulnerable to bias from initial training steps [Guu et al., 2017].

3.2 Intrinsic reward scaling and normalization

Since it is important that the intrinsic rewards at intermediate time steps of a trajectory must be less than the maximum extrinsic reward that is possible when $e_t = e_a$, we scale the intrinsic reward to

Table 1: Hits@N for our model CW vs $MINERVA$ for different training iterations

Hits	100_{CW}	200_{CW}	300_{CW}	400_{CW}	100_M	200_M	300_M	400_M
Hits@1	0.037	0.051	0.175	0.177	0.032	0.031	0.066	0.152
Hits@3	0.068	0.118	0.308	0.391	0.078	0.088	0.143	0.257
Hits@5	0.104	0.154	0.369	0.469	0.112	0.129	0.196	0.313
Hits@10	0.186	0.236	0.443	0.577	0.182	0.194	0.292	0.372
Hits@20	0.335	0.385	0.518	0.673	0.330	0.363	0.457	0.482

be between 0 and 1. The rewards are normalized to speed up learning "in the loop" by dividing the rewards by a running estimate of the standard deviation.

3.3 Combining extrinsic and intrinsic rewards

The extrinsic rewards are added to the normalized intrinsic rewards at all time steps. This would enforce that $R_{c,t} \in [0, 2]$ for all t. Cumulative discounted reward is calculated by using a single discount factor for both intrinsic as well as extrinsic rewards.

3.4 Incorporated methods

Several methods involved in training MINERVA [Das et al., 2018] were reused, such as the additive control variate baseline and entropy regularization term β.

4 Results

Our resulting model was tested on the Kinship dataset, which consists of a family tree of the Alyawarra tribe. Our testing procedure consisted of entity prediction of the form $(e_1, r, ?)$. The most probable destination entities from our trajectories were ranked using a beam search. The top 50 scoring entities are retained, based on the output logits of our model. The length of the trajectories were kept as 2.

Our experimental findings reveal that our model shows an uptick in average reward per batch around the 200th iteration as shown in Figure 1, which suggests that the model has started to explore more effective paths in the graph early on, leading to better route planning and transfer of knowledge.

Moreover, an increase in the Hits@n metric was observed as shown in Table 1. Considering the AUC-PR in this case is important as it is a proxy for the diversity of paths that CuriousWalk is able to find. AUC-PR represents the proportion of triples in the test set that can be found by the model with at least one path from the head entity to the tail entity. It is formally defined as:

$$\Sigma_{(e_q,r,e_a)\in D_{test}} Cnt(e_q, r, e_a)/|D_{test}|$$

where D_{test} is the number of examples in the test set, and $Cnt(e_q, r, e_a)$ is an indicator function to denote whether the model can find a path from h to t. The function value is 1 if at least one path can be found, otherwise, it is 0. The improvement, offers an insight into the robustness of the model, as shown in Figure 2.

5 Conclusion

In this study, we have explored the potential for an intrinsic reward structure to mitigate the sparsity and exploratory issues of MINERVA [Das et al., 2018]. The observed proficiency in the hits metric is noteworthy because it may signify that the denser reward structure is beneficial for effective learning. Furthermore, as training goes on the disparity between MINERVA and CuriousWalk implies a cumulative learning effect, resulting from the timely feedback from the environment. Diverse paths can enhance robustness of a reasoning model, and signifies adaptability in the case that certain paths are altered or become unavailable.

There are several limitations of intrinsic rewards to take into consideration for future research. Tuning the predictor network's as well as the agent's own hyperparameters can result in significant time

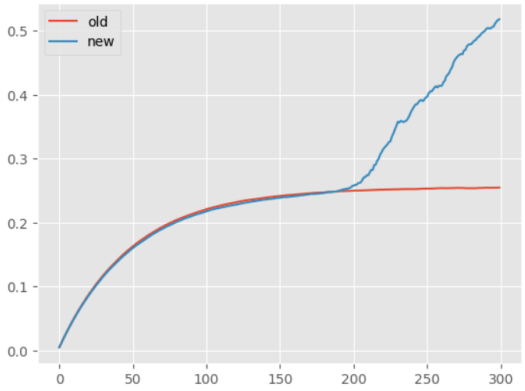

Figure 1: The upsurge in average reward per batch suggests better explorative performance of CuriousWalk. Note that this measurement does not include intrinsic rewards.

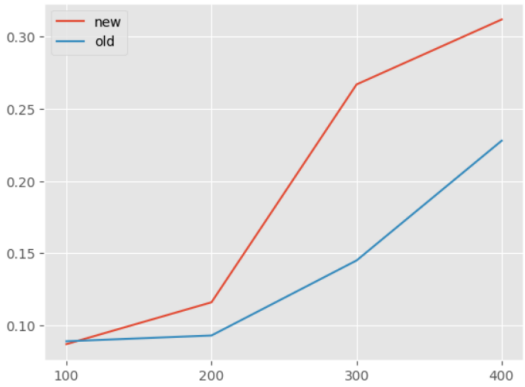

Figure 2: More AUC-PR indicates that in more number of testing examples, CuriousWalk could find at least one path from the query entity to the answer entity.

and computational overhead, and since the reasoning capabilities of our agent are sensitive to these hyperparameters, methods to make the search process easier should be investigated. On the other hand, the limited interpretability of our rewards can result in difficulty balancing exploration and exploitation. The agent can often prioritize novelty over task-specific goals like path finding.

There is scope for further experimentation with regards to testing the model on bigger datasets in the future, to understand how well CuriousWalk can generalize. Furthermore, the majority of existing literature in entity prediction use policy gradient methods to train the reasoner. However, off-policy methods are known to be more data efficient, especially since the environment is deterministic and transitions are quick. In this regard, Q-learning techniques with replay buffers may help improve performance.

References

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation, 2018.

Wenhu Chen, Wenhan Xiong, Xifeng Yan, and William Yang Wang. Variational knowledge graph reasoning. *NAACL*, 2018. doi: 10.18653/V1/N18-1165.

Rajarshi Das, Arvind Neelakantan, David Belanger, and Andrew McCallum. Chains of reasoning over entities, relations, and text using recurrent neural networks. *conference of the european chapter of the association for computational linguistics*, 2017. doi: 10.18653/V1/E17-1013.

Rajarshi Das, Shehzaad Dhuliawala, Manzil Zaheer, Luke Vilnis, Ishan Durugkar, Akshay Krishnamurthy, Alex Smola, and Andrew McCallum. Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning. In *International Conference on Learning Representations*, April 2018.

Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings, 2018.

Kelvin Guu, Panupong Pasupat, Evan Zheran Liu, and Percy Liang. From language to programs: Bridging reinforcement learning and maximum marginal likelihood, 2017.

Marcel Hildebrandt, Jorge Andres Quintero Serna, Yunpu Ma, Martin Ringsquandl, Mitchell Joblin, and Volker Tresp. Reasoning on knowledge graphs with debate dynamics, 2020.

Ni Lao, Tom Mitchell, and William W. Cohen. Random walk inference and learning in a large scale knowledge base. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 529–539, Edinburgh, Scotland, UK., July 2011. Association for Computational Linguistics. URL https://aclanthology.org/D11-1049.

Ruiping Li and Xiang Cheng. DIVINE: A generative adversarial imitation learning framework for knowledge graph reasoning. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2642–2651, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1266. URL https://aclanthology.org/D19-1266.

Xi Victoria Lin, Richard Socher, and Caiming Xiong. Multi-hop knowledge graph reasoning with reward shaping, 2018.

Xin Lv, Yuxian Gu, Xu Han, Lei Hou, Juanzi Li, and Zhiyuan Liu. Adapting meta knowledge graph information for multi-hop reasoning over few-shot relations, 2019.

Arvind Neelakantan, Benjamin Roth, and Andrew McCallum. Compositional vector space models for knowledge base inference. *national conference on artificial intelligence*, 2015.

Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction, 2017.

Yelong Shen, Jianshu Chen, Po-Sen Huang, Yuqing Guo, and Jianfeng Gao. M-walk: Learning to walk over graphs using monte carlo tree search, 2018.

Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction, 2016.

Guojia Wan and Bo Du. Gaussianpath: A bayesian multi-hop reasoning framework for knowledge graph reasoning. In *AAAI Conference on Artificial Intelligence*, 2021. URL https://api.semanticscholar.org/CorpusID:235306618.

Guojia Wan, Shirui Pan, Chen Gong, Chuan Zhou, and Gholamreza Haffari. Reasoning like human: Hierarchical reinforcement learning for knowledge graph reasoning. *IJCAI*, 2020. doi: 10.24963/IJCAI.2020/267.

Heng Wang, Shuangyin Li, Rong Pan, and Mingzhi Mao. Incorporating graph attention mechanism into knowledge graph reasoning based on deep reinforcement learning. *Conference on Empirical Methods in Natural Language Processing*, 2019. doi: 10.18653/V1/D19-1264.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3–4):229–256, may 1992. ISSN 0885-6125. doi: 10.1007/BF00992696. URL https://doi.org/10.1007/BF00992696.

Wenhan Xiong, Thien Hoang, and William Yang Wang. Deeppath: A reinforcement learning method for knowledge graph reasoning, 2018.

Denghui Zhang, Zixuan Yuan, Hao Liu, Xiaodong Lin, and Hui Xiong. Learning to walk with dual agents for knowledge graph reasoning, 2021.