Table of Contents

Enhanced DeepPath

An Enhanced Reinforcement Learning Method for Knowledge Graph Reasoning

Knowledge Graph

Semantic Networks were invented to address the growing need for a knowledge representation framework that can capture a wide range of entities — real-world objects, events, situations or abstract concepts and relations.

Every Company/Group/Individual creates their own version of the Knowledge Graph to limit complexity and organize information into data and knowledge such as Google's Knowledge Graph. But Knowledge Representation brings the ability to represent entities and relations with high reliability, explainability, and reusability. [1]

Knowledge graphs can be used in QA, recommender systems, search engines, etc.

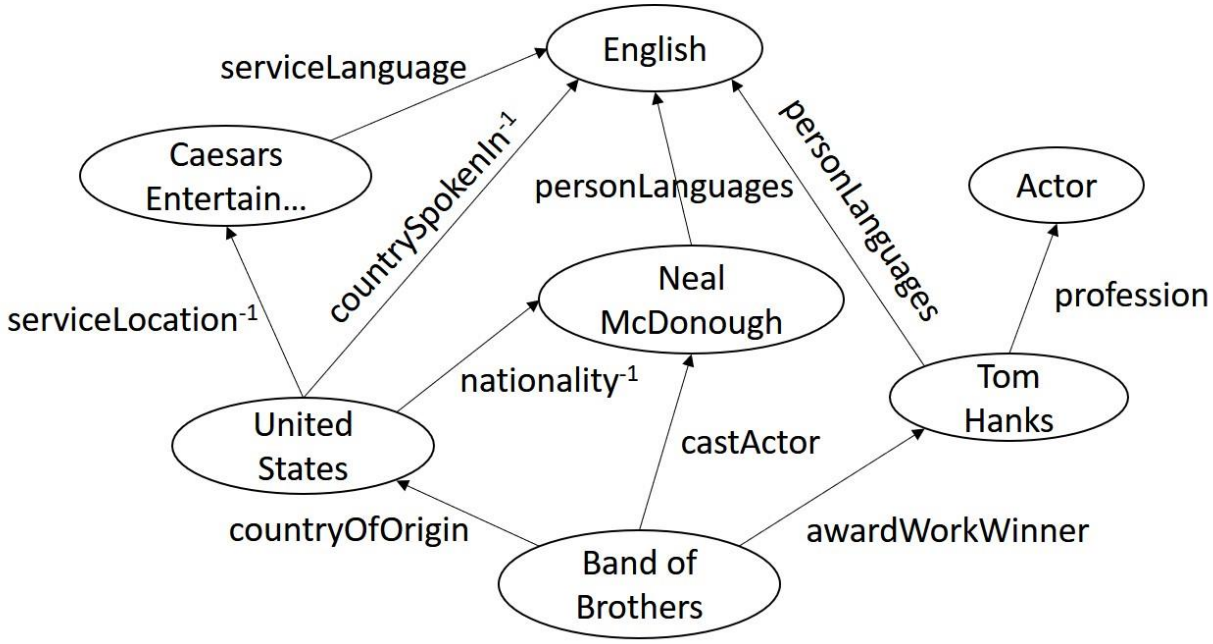

Figure 1 Knowledge Graph

Reasoning on Knowledge Graph

What we are dealing with is the triples of the entity-relations in form of *subject-predicate-object*. Reasoning in KG is the ability to calculate the set of triples that logically follow from a knowledge graph and a set of rules. Such logical consequences are materialized as new triples in the graph. [2]

Consider following rules:

- Berlin is in Germany
- Germany is in Europe

As the relation in is intuitively transitive, then we can obtain a rule such as *Berlin is in Europe*. However, the triplet *Berlin is in Europe* is missing from KG. We can obviously add this rule by hand to complete graph, but these removes some benefits such as:

- Rules are too many and this task manually can be cumbersome
- We are not using the implicit logic of being transitive for rules such as in.

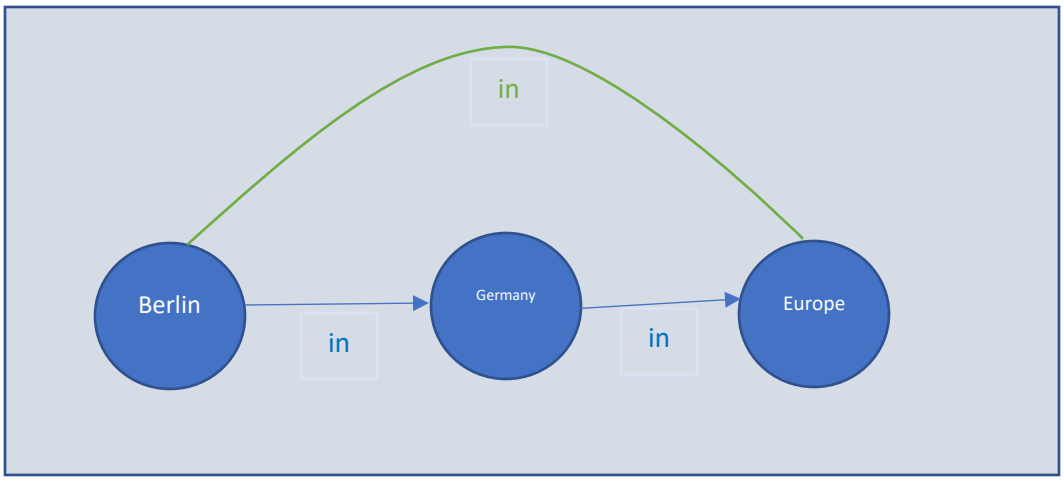

Figure 2 Reasoning on KG

Datasets

There are two datasets that has been used in this experiment. Both of these datasets are subset of larger datasets.

Dataset	# of Entities	# of Relations	# of Triples	# of Tasks
FB15k-237[3]	14,505	237	310,116	20
NELL-995	75,492	200	154,213	12

Figure 3 Datasets

FB15k-237: Constructed from FB15k (Bordes et al., 2013), redundant relations removed

NELL-995: Constructed from the 995th iteration of NELL system (Carlson et al., 2010b)

For both of these datasets, following processing steps have been incorporated:

- Remove useless relations: *haswikipediaurl*, *generalizations, …*
- Add inverse relation links to the knowledge graph (-1 superscripts)
- Remove the triples with task relations

They perform the reasoning tasks on 20 relations which have enough reasoning paths. These tasks consist of relations from different domains like Sports, People, Locations, Film, etc.

To facilitate path finding, they also add the inverse triples. For each triple (h, r, t), they append $(t, r-1, h)$ to the datasets. With these inverse triples, the agent is able to step backward in the KG.

For each *reasoning task* r_i , we remove all the triples with r_i or r_i^{-1} from the KG. These removed triples are split into train and test samples.
For the *link prediction task*, each h in the test triples $\{(h, r, t)\}$ is considered as one query. A set of candidate target entities are ranked using different methods. For fact prediction, the true test triples are ranked with some generated false triples.

DeepPath

In this section, the different aspects of DeepPath [4] paper will be discussed as the core of adopting RL in KG reasoning. In summary, the proposed approach can be expressed by following:

- Learning the paths instead of using random walks
- Model the path finding as an MDP
- Train a RL agent to find paths
- Using a reward function that considers accuracy, efficiency, and diversity simultaneously
- Use the learned paths as horn clauses

Related Works

Related works in this area can be dichotomized into two main categories:

1. Path based methods: Which usually find potential path types between entities pairs then compute rank of paths using Random Walk. The main issues with these approaches are that in the dense area of graph where a node is connected to a large number of nodes, fan out dominates the algorithm and speeds decreases drastically.
2. Embedding based methods: The idea is that the h vector + r should be close to t vector. This has been incorporated into loss function. Some of these models create too many models which make them fail to scale or even cannot model the relational path explicitly.

Note that the RL approach in DeepPath, uses embedding from TransE to embed the entities with size of 200 where is the input size of policy network too.

Reinforcement Learning

The RL system consists of two parts; The first part is the external environment E which specifies the dynamics of the interaction between the agent and the KG. This environment is modeled as a Markov decision process (MDP).

A tuple $\{S, A, P, R\}$ is defined to represent the MDP where S, A, P and R are state space, action space, transition probability and rewards respectively. The important property in this section is that the state space S is continuous which has been constructed from KG embeddings such as TransE.

The second part of the system, the RL agent, is represented as a policy network which maps the state vector to stochastic policy. The NN parameters θ are updated using stochastic gradient descent in $\pi_\theta(\mathrm{s}, \mathrm{a}) = p(a \mid s;\ \theta)$. This config enables agent to learn much complex paths where greedy policies can fail.

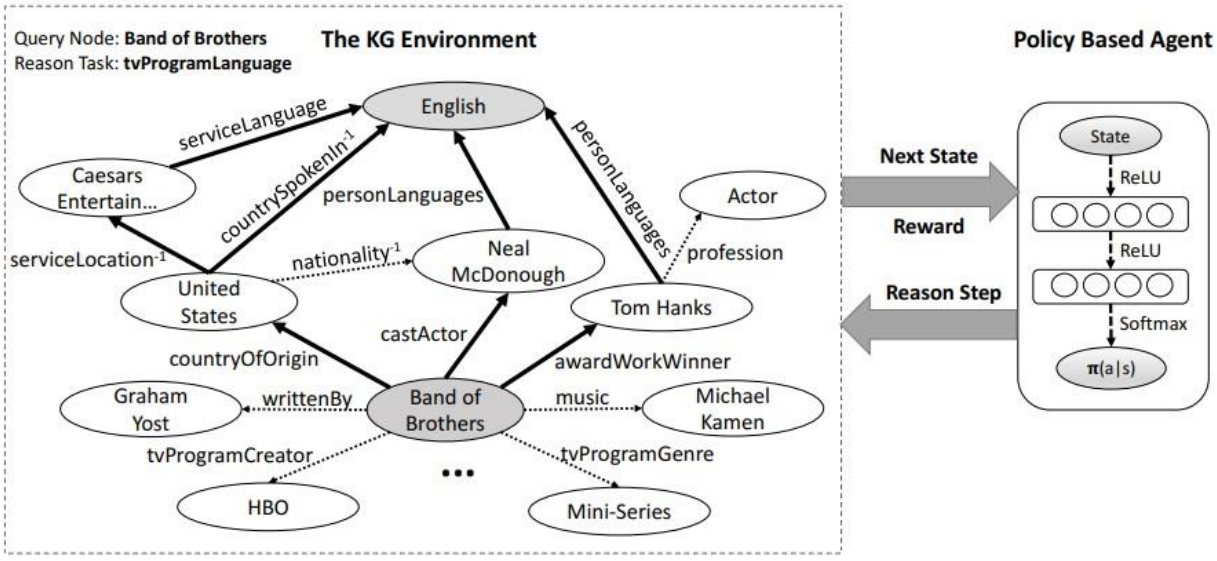

Figure 4 RL model.

On the left side, the KG environment ε modeled by an MDP. The dotted arrows (partially) show the existing relation links in the KG and the bold arrows show the reasoning paths found by the RL agent. On the right side, the structure of the policy network agent can be seen. At each step, by interacting with the environment, the agent learns to pick a relation link to extend the reasoning paths.

Actions

Given the entity pairs (e_s, e_t) with relation r, we want the agent to find the most informative paths linking these entity pairs.

States

It is impossible to discreetly model all entities and relations which are discrete in first place. To overcome this issue, they use translation embeddings such as TransE to represent entities and relations. This embedding map all symbols into low dimensional vectors. In this model, each state holds the agent's position in the environment ε. The destination and source entities (states) are linked using relations (actions). For step t we have $S_t = (e_t, e_{target}\text{-}e_t)$ where e_x denotes the embedding of entity.

Reward Function

Three factors incorporate in the defined reward function to achieve accuracy, efficiency and diversity which are as follow:

- Global accuracy: As the KG can be enormously big, and due to that, agent makes many mistakes w.r.t. correct decisions which can be increased exponentially, they introduced binary outcome of success or failure.

$$r_{global} = \begin{cases} +1 & ; \text{ if reaches the target} \\ -1 & ; \text{ o.w.} \end{cases}$$

- Path efficiency: It has been observed that shorter paths have more intuitive logics so limiting the amount of agent's interaction with environment can help where $path$ is a sequence of relations.

$$r_{efficiency} = \frac{1}{length(path)}$$

- Path Diversity: The agent tends to find paths with similar syntax and semantics, which usually have high correlations because mostly are redundant. To ensure diversity, they introduce a diversity functions based on cosine similarity functions to enforce agent to learn paths far (higher distance) from what it has learned so far.

$$r_{\text{diversity}} = -\frac{1}{|F|}\sum_{1}^{|F|}\cos(p, p_i)$$

PS. Here is the section some improvements have been introduced by using dropout to forcing diversity.

Policy Network

They use a fully-connected neural network to parameterize the policy function $\pi(s;\ \theta)$ that maps the state vector s to a probability distribution over all possible actions. The neural network consists of two hidden layers, each followed by a rectifier nonlinearity layer (ReLU). The output layer is normalized using a softmax function.

PS. Here is the section some improvements have been introduced by using dropout to forcing diversity.

Challenge

A big issue with RL models is that when search space increases exponentially, models fails to converge. This also can happen in this task as knowledge graph can have more than hundreds of possible valid actions. To tackles this issue, they have adopted the same approach used in *AlphaGo* (Silver et al., 2016) paper. The idea is to start training by providing a supervision signal to help model as a teacher to choose wiser. In this article, the supervised policy is trained with randomized breath-first-search (BFS).

Supervised (Imitation) Policy Learning

For each relation, they use a subset of all the positive samples (entity pairs) to learn the supervised policy where a two-side BFS has been conducted to find the correct path, where for each $path$, they update parameters θ to maximize the expected cumulative reward using Monte-Carlo Policy Gradient. This can be showed in this way:

$$\begin{aligned}\nabla_\theta J(\theta) &= \sum_t \sum_{a\in\mathcal{A}} \pi(a|s_t;\theta)\nabla_\theta \log \pi(a|s_t;\theta) R(s_t, a_t) \\ &\approx \nabla_\theta \sum_t \log \pi(a = r_t|s_t;\theta) R(s_t, a_t)\end{aligned}$$

Figure 5 Supervised policy learning

$$R(s_t, a_t) = \lambda_1 r_{global} + \lambda_2 r_{efficiency} + \lambda_3 r_{diversity}$$

Figure 6 Combined Reward Function

But there is issue that incorporated BFS tends to find shorter paths which is biased, but we would like to find paths based on their given reward only not the bias of supervision. The idea is to retrain the supervised policy network with rewards.

For each relation, the reasoning with one entity pair is treated as one episode which picks different relations. This relation link may lead to a new entity, or it may lead to nothing. These failed steps will

cause the agent to receive negative rewards. The agent will stay at the same state after these failed steps. Since the agent is following a stochastic policy, the agent will not get stuck by repeating a wrong step but its repetition has been limited to a max bound to increase efficiency. After each episode, the policy network is updated using the aforementioned gradient and reward function. (*See algorithm 1 in the paper*)

To show the effect of the supervised training, we evaluate the agent's success ratio of reaching the target within 10 steps ($succ_{10}$) after different number of training episodes. For each training episode, one pair of entities (e_{source}, e_{target}) in the train set is used to find paths.

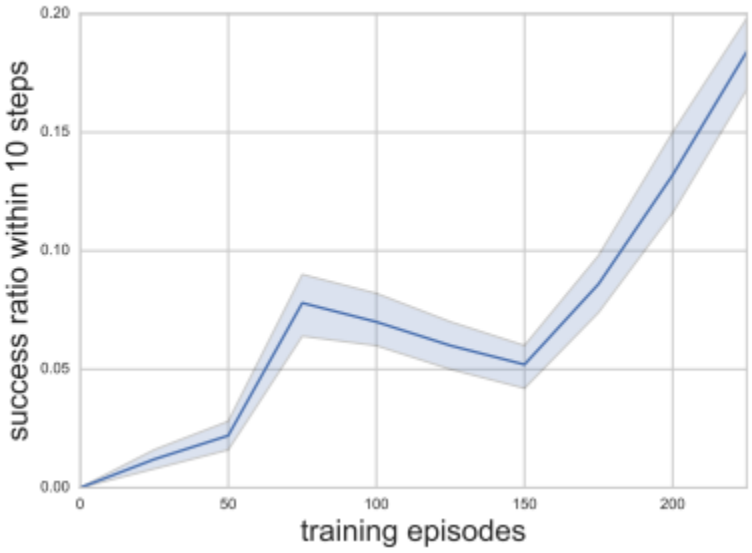

Figure 7 The success ratio (succ10) during training.
task: athletePlaysForTeam

Inference Using Learned Paths

Given an entity pair, the reasoning paths learned by the RL agent can be used as logical formulas to predict the relation link where each formula can be verified using a bi-directional search. The idea is that in a KG, an entity node can be linked to many other nodes with the same relation link where number of intermediate nodes may increase exponentially which leads to poor convergence. However, if the formula being verified from the inverse direction, the number of intermediate nodes can be decreased massively. (*See algorithm 2 in the paper*)

Simply the idea is to check that the formulas hold for entity pairs in both directions.

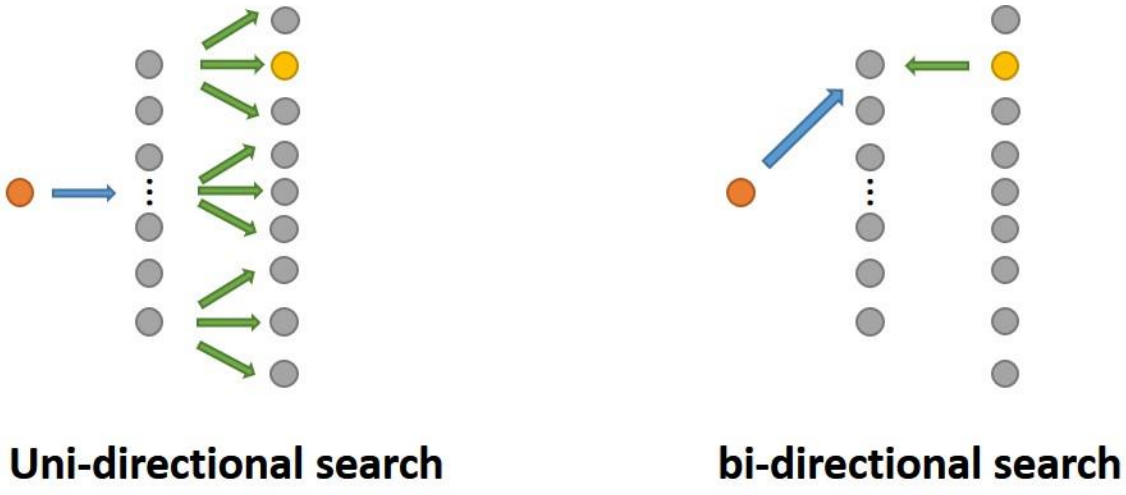

Figure 8 Inference using Learned Paths

Results

To evaluate the reasoning formulas found by their RL agent, they explore two standard KG reasoning tasks: link prediction (predicting target entities) and fact prediction (predicting whether an unknown fact holds or not). They compare our method with both path-based methods and embedding based methods. Finally, we conduct an experiment to investigate the effect of the supervised learning procedure.

In this section we just some see metrics upon experimenting the proposed approach on given dataset NELL-995:

Tasks	PRA	Ours	TransE	TransR
worksFor	0.681	**0.711**	0.677	0.692
atheletPlaysForTeam	**0.987**	0.955	0.896	0.784
athletePlaysInLeague	0.841	**0.960**	0.773	0.912
athleteHomeStadium	0.859	**0.890**	0.718	0.722
teamPlaysSports	0.791	0.738	0.761	**0.814**
orgHirePerson	0.599	**0.742**	0.719	0.737
personLeadsOrg	0.700	**0.795**	0.751	0.772
Overall	0.675	**0.796**	0.737	0.789

Figure 9 Link Prediction Result MAP

For Qualitative Analysis, to illustrate the effect of the efficiency reward function, they show the path length distributions and the extracted rules as follows:

personNationality:

- placeOfBirth -> locationContains^{-1}
- peoplePlaceLived -> locationContains^{-1}
- peopleMariage -> locationOfCeremony -> locationContains^{-1}

tvProgramLanguage:

- tvCountryOfOrigin -> countryOfficialLanguage
- tvCountryOfOrigin -> filmReleaseRegion^{-1} -> filmLanguage
- tvCastActor -> personLanguage

Figure 10 Example Reasoning Paths Found by RL

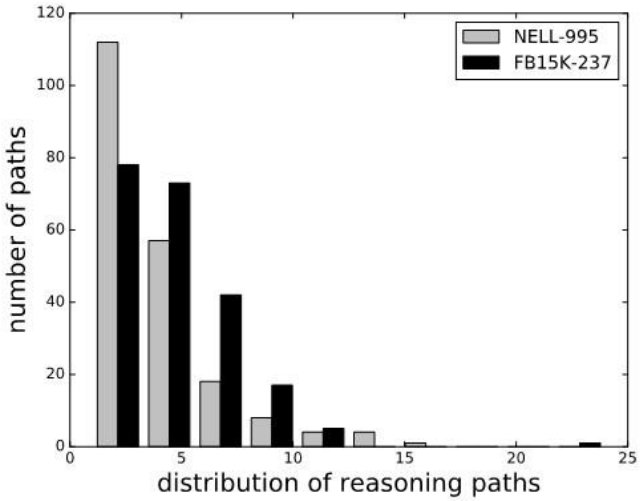

Figure 11 The distribution of paths lengths on two datasets

Another important observation is that their model uses much fewer reasoning paths than PRA, which indicates that our model can actually extract the most reliable reasoning evidence from KG.

Tasks	# of Reasoning Paths	
	PRA	RL
worksFor	247	25
teamPlaySports	113	27
teamPlaysInLeague	69	21
athletehomestadium	37	11
organizationHiredPerson	244	9
...		
Average #	137.2	20.3

Figure 12 Number of reasoning paths used by PRA and their RL

Enhancement

The introduced enhancements are defined to work on the RL framework to just improve the path finding algorithm better. All other sections are similar to the previous framework and the only differences have been described in the following sections which exactly matches the previously defined framework DeepPath. These ideas completely have been adopted from Salesforce's paper "Multi-hop knowledge graph reasoning learned via policy gradient with reward shaping and action dropout" [5].

Reinforcement Learning

We can use language embedding and other pretrained models to enhance our agent's behavior. To do so, we can incorporate path finding score as part of reward or in term of policy network, we can explicitly delete some relations to enforce agent's action to be exposed to more diversity by making more exploration which can be achieved by using dropouts in the defined policy neural network. You can find the changes in more details in the following sections.

Reward Function

the agent receives a binary reward based on solely the observed answers. However, it is intrinsically incomplete and this approach rewards the false negative search results identically to true negatives. To alleviate this problem, we use existing KG embedding models designed for the purpose of KG completion *(Trouillon et al., 2016; Dettmers et al., 2018)* to estimate a soft reward for target entities whose correctness is unknown.

Finally, the embedding method uses the corresponding vector space to estimate likelihood.

$$R(s_T) = R_b(s_T) + (1 - R_b(s_T))f(e_s, r_q, e_T)$$

Figure 13 Reward shaping strategy

Namely, if the destination e_t is a correct answer, the agent receives reward 1. Otherwise the agent receives a fact score estimated by $f(e_s, r_q, e_t)$, which is pre-trained.

Policy Network

Formally, the REINFORCE algorithm has been used. The issue is that if agent reaches to the correct target but using wrong inference and relation link, as there is no oracle to validate this, it may cause agent to learn completely wrong rules because the path is irrelevant to the query.

Since there are usually more spurious paths than correct ones, spurious paths are often found first, and following exploration can be increasingly biased towards them.

Here they propose the action dropout technique which randomly masks some outgoing edges for the agent in the sampling step of REINFORCE. The agent then performs sampling according to the adjusted action distribution of Bernouli uniform distribution.

$$\tilde{\pi}_\theta(a_t|s_t) \propto (\pi_\theta(a_t|s_t) \cdot \mathbf{m} + \epsilon)$$
$$m_i \sim Bernoulli(1-\alpha), i = 1, \ldots |A_t|$$

Figure 14 e Adjusted action distribution

Where m is a binary variable sampled from the Bernoulli distribution with parameter $1 - \alpha$. A small value ε is used to smooth the distribution in case $m = 0$.

Overall approach can be seen here:

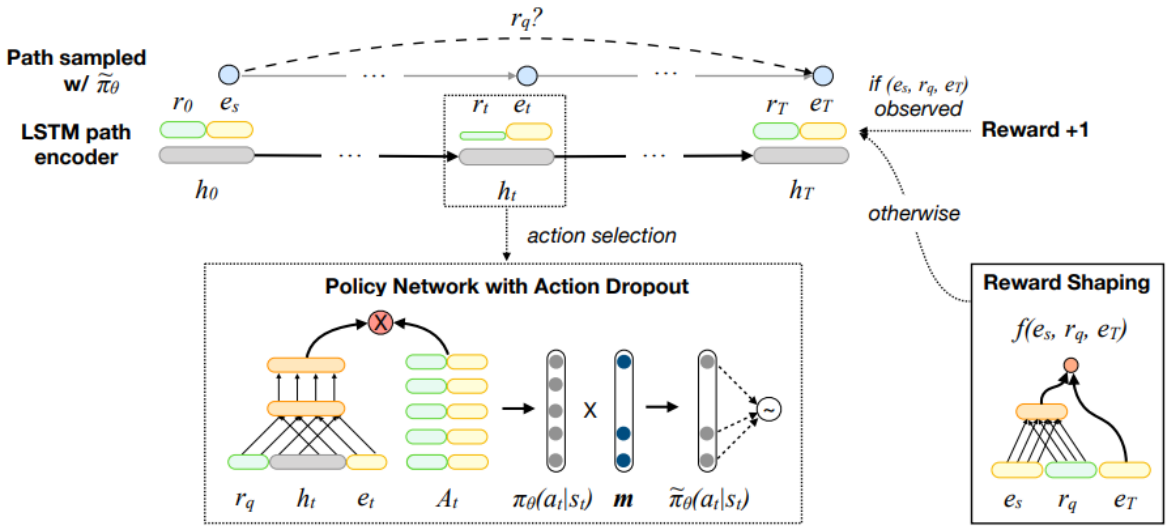

Figure 15 Reward Shaping and Action Dropout Overview

At each time step t, the agent samples an outgoing link according to $\tilde{\pi}\theta(a_t|s_t)$, which is the stochastic REINFORCE policy $\pi\theta(a_t|s_t)$ modified by a random binary mask m. The agent receives reward 1 if stopped at an observed answer of the query $(e_s, r_q, ?)$; otherwise, it receives reward $f(e_s, r_q, e_t)$ estimated by the reward shaping (RS) network. The RS network is pre-trained and doesn't receive gradient updates.

References

[1] Sudip Chowdhury, Knowledge Graph: The Perfect Complement to Machine Learning, 2019, medium.

[2] Peter Crocker, The intuitions behind Knowledge Graphs and Reasoning, 2018, Towards Data Science.

[3] Toutanova et al. Representing text for joint embedding of text and knowledge bases

[4] Xiong, Wenhan, Thien Hoang, and William Yang Wang. "Deeppath: A reinforcement learning method for knowledge graph reasoning." arXiv preprint arXiv:1707.06690 (2017).*

[5] Lin, Xi Victoria, Richard Socher, and Caiming Xiong. "Multi-hop knowledge graph reasoning with reward shaping." arXiv preprint arXiv:1808.10568 (2018).*

PS. References with * are main papers.

Appendix

- Link to DeepPath source code: https://github.com/xwhan/DeepPath
- Link to salesforce source code: https://github.com/salesforce/MultiHopKG
- Link to my source code: https://github.com/Nikronic/DeepPath

Acknowledgment

There is nothing new in my source code, all I have done is first fixed some errors and issues regarding Python 2.7 and older version of Tensorflow and merged the idea of section “enhancement” from second source code into first one. So, if there is any improvement in results, all credits belong to the real authors, I have just used theirs and constructed new blocks without mathematical verification.