Nonparametric Contextual Reasoning for Question Answering over Knowledge Bases

Thesis Document

Rajarshi Das
College of Computer and Information Sciences
University of Massachusetts Amherst
rajarshi@cs.umass.edu

Advisor: Andrew McCallum

Committee: Mohit Iyyer, Marco Serafini, Kyunghyun Cho, Chris Dyer

Draft Date: February 15, 2022

Abstract

Question answering (QA) over knowledge bases provides an user-friendly access to the information stored in them. We have experienced a tremendous progress in the performance of QA systems, thanks to the recent advancements in representation learning by deep neural models. However, such deep models function as black-boxes with an opaque reasoning process, are brittle, and offer very limited control (e.g. for debugging an erroneous model prediction). It is also unclear how to reliably add or update knowledge stored in their model parameters.

This thesis proposes nonparametric models for question answering that disentangle logic from knowledge. For a given query, the proposed models are capable of deriving interpretable reasoning patterns "on-the-fly" from other contextually similar queries in the training set. We show that our models can seamlessly handle new knowledge (new entities and relations) as they are continuously added to the knowledge base. Our model is effective for complex and compositional natural language queries requiring subgraph reasoning patterns and work even when annotations of the reasoning patterns (logical forms) are not available, achieving new state-of-the-art results on multiple benchmarks. Leveraging our nonparametric approach, we also demonstrate that it is possible to correct wrong predictions of deep QA models without any need for re-training, thus paving a way towards building more controllable and debuggable QA systems. Finally, compared to deep parametric models, this thesis demonstrates that nonparametric models of reasoning (i) can generalize better to questions needing complex reasoning especially when the number of questions seen during training are limited (ii) can reason more effectively as new data is added, (iii) offer more interpretability for its prediction and (iv) are more controllable and debuggable.

Contents

1 Introduction

Automated reasoning, the ability of computing systems to make new inferences from observed evidence, has been a long-standing goal of artificial intelligence. Knowledge bases (KBs) both automatically and manually constructed, are often incomplete. However, many valid unobserved facts can be inferred from observed KB facts by *reasoning* over them. We are interested in automated reasoning on large KBs with rich and diverse semantic types (Suchanek et al., 2007; Bollacker et al., 2008; Carlson et al., 2010). Much of the information stored in KBs are symbolic facts of the form $(\mathrm{e}_1, \mathrm{r}, \mathrm{e}_2)$, where e_1, e_2 denote entities and r denotes a semantic relation. KBs can be naturally described as a graph where the entities are nodes and the relations are labelled edges. An effective and user-friendly way of accessing the information stored in a KB is by issuing queries to it. Such queries can be structured (e.g. queries for booking flights) or unstructured (e.g. natural language queries). A challenge for question answering (QA) systems over KBs is to handle queries whose answers are not directly stored (as a simple fact) in the KB and instead the QA model needs to reason in order to derive the answer from other observed facts. This thesis focuses on building QA systems over structured KBs that can perform such reasoning.

We have experienced tremendous progress in the performance of QA and KB completion systems, thanks to the recent advancements in representation learning by deep neural models. However, such deep models also come with a lot of practical deficiencies. For example,

- Current QA and KB completion models are often black-box scoring functions that learn vector representation of a fixed vocabulary of entities and relations (Bordes et al., 2013; Socher et al., 2013; Sun et al., 2019b; Gu et al., 2021). The parameters of the model store both the logic and knowledge making the underlying reasoning process that the models use, opaque and unclear.
- There are essentially unbounded types of questions that can be asked to a QA system and hence the type of reasoning patterns that a model has to learn and store in its parameters are massive. Moreover during training, a model might encounter only a few training examples for each question type. We show that parametric models for KBQA struggle in such settings (Das et al., 2022).
- We live in an evolving world with a lot of heterogeneity alongside which new entities and relations are being continuously created. For example, scientific papers and Wikipedia pages describing facts about new entities are being constantly added. These new findings further trigger the inference of newer facts, each with its own diverse reasoning. Because of the fixed pre-defined vocabulary, current models of KB completion cannot handle newly added entities and relations and cannot reason with new data.
- As we observe new evidence (data), the reasoning process of humans become more nuanced., i.e. new reasoning rules appear and existing reasoning rules evolve with the addition of new data. It is unclear how to robustly update the parameters of deep neural models to encode the updated (and new reasoning rules). As we show later in the thesis, models when trained on newly arriving data tend to suffer from catastrophic forgetting, where the model has forgotten what it had learned before (Das et al., 2020b, 2021)
- Finally, when models output a wrong prediction for a given question, they offer us very limited insights and control for *debugging* the wrong prediction (Das et al., 2021). We show that nonparametric models allows us new opportunities to fix erroneous model predictions by letting us "inject" cases to the KNN memory, making the models more *interpretable* and *controllable*.

1.1 Desiderata

We hope to design accurate models for question answering over large KBs with the following desirable properties:

- **Generalize to newly arriving data**: We need models that can seamlessly reason with the dynamic world. This means that the models should be able to handle new entities and relationships as well as new (latent) reasoning rules that emerge as new data is added.

- **Accurately answer diverse natural language queries that need complex reasoning patterns**: Natural language interface to KBs provide an user-friendly access to the information stored in them. Natural language queries also allow us to ask more complex questions that are difficult to be expressed in a single KB relation. Moreover, seemingly simple questions can require complex reasoning involving multiple facts in the KB (e.g. How many Nobel prizes did the members of the Curie family win?). The reasoning patterns required to answer many such questions take the shape of subgraphs and are often more complex than simple reasoning chains. We need models that can perform the required reasoning over complex (latent) subgraph pattens to accurately answer such questions.
- **Interpretable Reasoning Process with Controllability and Debuggability**: Current QA and KB completion models function as blackboxes offering limited insights into their reasoning process. Moreover, when they output a wrong prediction for a query, there is nothing much we can do other than re-training the model on the failed inputs.We need models that not only offer more transparency in how they arrive at an answer to a query but also are controllable and debuggable, giving better insights into how to fix erroneous model predictions.

1.2 Summary of Completed Work

This thesis proposes to build models that *disentangle* logic from data stored in the model parameters. Instead of memorizing the data and storing it in the parameters, the proposed model learns how to navigate the knowledge graph to answer a given query. Our proposed solution MINERVA (Das et al., 2018), treats the knowledge graph (KG) as an environment in a reinforcement learning (RL) framework. Starting from the node corresponding to the query entity, MINERVA traverses the KG by selecting an outgoing edge from the set of all edges (action space). This is repeated till the agent decides to stop at a node that it thinks as the answer. During training, the environment gives a positive reward if the terminal node is the answer to the given query. It should be emphasized that MINERVA, instead of memorizing the KG, learns to navigate it for a given query. This allows MINERVA to generalize to new parts of the graph containing *unseen entities*. Additionally, the sequence of relational edges traversed also offers interpretability to the reasoning process adopted by the model. However, the logic to navigate the KG, i.e. which relational edge to choose next, is entirely encoded in the parameters of the model. This implies that MINERVA will not be able to adapt to *new relations* added to the KG as there will be no trained parameters for them. Also, as new information is added, existing rules evolve and new reasoning rules emerge. MINERVA will not be able to use the new (or evolved) rules for reasoning (without re-training) as new data is added.

To be able to reason with the dynamic world, we propose models that are capable of doing *nonparametric* and *contextual reasoning* (Das et al., 2020a). Given a query about a new entity, our model finds other contextually similar entities in the KG for which we observe the query relation. Next, we gather multiple reasoning paths that connect the retrieved entities to the entities they are connected to using the query relation. Lastly, these reasoning paths are then traversed in the subgraph around the query entity to derive the answer. This nonparametric approach allows us to seamlessly reason with newly arriving data. Reasoning patterns required for answering a query about a newly added entity can be derived from existing entities in the KB. Similarly, any newly added data can be incorporated to refine the prediction of a query about an existing entity. The algorithm proposed in Das et al. (2020a) treated all the paths gathered from the contextual entities equally. However, not all paths are equally important for reasoning. In fact, there often exists "spurious" paths which connect the question entities to the answer entity, but are not valid chains of reasoning. To deal with this scenario, we propose a probabilistic approach that weighs each reasoning path with its prior and precision scores (Das et al., 2020b). We demonstrate that our simple CBR-based approach not only achieves state-of-the-art performance on various KB completion benchmarks (Toutanova and Chen, 2015; Xiong et al., 2017), but also outperforms SOTA models by a wide margin on a challenging open-world KB completion setup, where new entities and facts are being continuously added.

The work described in this thesis till now, accepts structured queries $(\mathrm{e}_1, \mathrm{r}, ?)$ as input. As previously mentioned, natural language interfaces to KBs provides a user-friendly access to the information stored in the KB. Natural language (NL) queries also allow us to ask questions which are hard to be expressed with a single KB relation. To handle complex NL queries, we introduce a nonparametric approach for semantic

parsing that retrieves other similar NL queries and their logical forms from the training set. Then a neural encoder-decoder model conditions on the retrieved cases to derive the logical form for the given query (Das et al., 2021). We show that our proposed approach is highly effective for complex compositional questions that require novel combination of KB relations not seen during training, thereby achieving state-of-the-art results in multiple KBQA datasets (Yih et al., 2016; Talmor and Berant, 2018; Keysers et al., 2020).

The approaches described above are reminiscent of a nonparametric framework proposed decades ago in classical AI — case-based reasoning (Schank, 1982; Kolodner, 1983). A sketch of a CBR system (Aamodt and Plaza, 1994) comprises of — (i) a retrieval module, in which 'cases' that are similar to the given problem are retrieved, (ii) a reuse module, where the solutions of the retrieved cases are re-used to synthesize a new solution. Often, the new solution does not work and needs more revision, which is handled by (iii) a revise module. Recently, there has been a lot of work in QA that retrieve relevant evidence from a nonparametric memory (Guu et al., 2020; Lewis et al., 2020b; Karpukhin et al., 2020). Another line of work (Lewis et al., 2021) pre-generates a large set of questions from documents. Answering a question boils down to finding the most similar question paraphrase in the generated list (using KNN search) and returning the corresponding answer.

Our CBR approach, in contrast, retrieves a contextually similar query (or an entity) from the training data (or KG) w.r.t the given query. The retrieved query need not be a paraphrase but should have relational similarity (e.g. for the query "Which countries border United States?", a nearest neighbor question could be "Which countries border India?") Next, it finds patterns (KG paths or subgraphs) that explains the reasoning behind the solution to the retrieved queries. Finally, the retrieved reasoning patterns are used to derive a reasoning pattern that works for the given query. In comparison to the method proposed by Lewis et al. (2021),we believe our approach (Das et al., 2021) is a more robust way of doing nonparametric reasoning as there is no guarantee all questions can be pre-generated from a document.

The previous work of Das et al. (2021), however, needed annotations of logical forms during training which is very expensive to annotate at scale. Moreover, the annotations for one schema do not generalize to a different schema. We hypothesize in a large KB, reasoning patterns required to answer a query type reoccur for various entities in their respective subgraph neigborhoods (Das et al., 2022). Leveraging this structural similarity between local neighborhoods of different subgraphs, we introduce a semiparametric model with (i) a nonparametric component that for each query, dynamically retrieves other similar k-nearest neighbor (KNN) training queries along with query-specific subgraphs and (ii) a parametric component that is trained to identify the (latent) reasoning patterns from the subgraphs of KNN queries and then apply it to the subgraph of the target query. We also propose a novel algorithm to select a query-specific compact subgraph from within the massive knowledge graph (KG), allowing us to scale to full Freebase KG containing billions of edges.

Lastly, this thesis explores another useful property of nonparametric models. When a QA model outputs an incorrect prediction for a query, nonparametric models give us the opportunity to 'fix' the wrong prediction by injecting relevant cases to the KNN index. The other alternative is to train the model on failed inputs, but we find that models often exhibit *catastrophic forgetting* unless carefully fine-tuned on the failed examples. Instead, we show that on addition of few simple cases (query, logical form pair) to the KNN index, our model is able to retrieve the newly added cases and use it to derive the correct logical form, essentially fixing a wrong prediction. We take advantage of this property to demonstrate that a QA model can be made to answer queries that need relation, the model has never been trained on, paving a way towards practical production-ready models.

1.3 Document Organization

This thesis proposal document is organized as follows. Chapter 2 introduces our proposed model (MINERVA) that decouple logic from knowledge in model parameters. We will demonstrate how MINERVA walks on the KG to find the answer to a given query. In chapter 3, we describe our nonparametric reasoning approaches to KB completion and demonstrate its efficacy on a challenging open-world knowledge graph completion setting. Chapter 4 extends our nonparametric approach to complex compositional queries in natural

language. Taking advantage of the nonparametric property of our model, we show that it is possible to "fix" wrong predictions of deep QA models, without re-training them. Finally, chapter 5 describes our solution for doing KBQA for questions needing complex subgraph reasoning patterns *without* needing annotated logical forms during training. We show that our proposed model (CBR-SUBG) can answer questions requiring complex reasoning patterns even though it sees only few examples of the question types during training; can generalize to graphs containing completely new entities and outperforms path-based models convincingly. We also show that the model performance, in true nonparametric fashion, increases as more evidence (similar questions) are retrieved.

This proposal describes work from the following research papers.

- **Rajarshi Das**, Shehzaad Dhuliawala, Manzil Zaheer, Luke Vilnis, Ishan Durugkar, Akshay Krishnamurthy, Alex Smola, and Andrew McCallum. 2018. Go for a walk and arrive at the answer: Reasoning over pathsin knowledge bases using reinforcement learning. In ICLR
- **Rajarshi Das**, Ameya Godbole, Shehzaad Dhuliawala, Manzil Zaheer, and Andrew McCallum. 2020a. A simple approach to case-based reasoning in knowledge bases. In AKBC (**Best paper runner up**)
- **Rajarshi Das**, Ameya Godbole, Nicholas Monath, Manzil Zaheer, and Andrew McCallum. 2020b. Probabilisticcase-based reasoning for open-world knowledge graph completion. In EMNLP findings
- **Rajarshi Das**, Manzil Zaheer, Dung Thai, Ameya Godbole, Ethan Perez, Jay-Yoon Lee, Lizhen Tan, LazarosPolymenakos, and Andrew McCallum. 2021. Case-based reasoning for natural language queries overknowledge bases. EMNLP
- **Rajarshi Das**, Ameya Godbole, Ankita Naik, Elliot Tower, Manzil Zaheer, Robin Jia, Hannaneh Hajishirzi, Andrew McCallum. 2022. Semiparametric Subgraph Reasoning for Question Answering over Large Knowledge Bases. In Submission. ICML

Following works are not included in the thesis but they have definitely played a very important role in formation of the main ideas presented in the thesis

- **Rajarshi Das**, Arvind Neelakantan, David Belanger, and Andrew McCallum. 2017. Chains of reasoning overentities, relations, and text using recurrent neural networks. In EACL.
- **Rajarshi Das**, Manzil Zaheer, Siva Reddy, Andrew McCallum. 2017. Question Answering on Knowledge Bases and Text using Universal Schema and Memory Networks. In ACL
- **Rajarshi Das**, Shehzaad Dhuliawala, Manzil Zaheer, Andrew McCallum, Shehzaad Dhuliawala, Manzil Zaheer, Andrew McCallum. 2019. Multi-step Retriever-Reader Interaction for Scalable Open-domain Question Answering. In ICLR
- **Rajarshi Das**, Tsendsuren Munkhdalai, Eric Xingdi Yuan, Adam Trischler, Andrew McCallum. 2019. Building Dynamic Knowledge Graphs from Text using Machine Reading Comprehension. In ICLR

2 Decoupling Logic from Knowledge stored in Model Parameters

2.1 Introduction

Automated reasoning, the ability of computing systems to make new inferences from observed evidence, has been a long-standing goal of artificial intelligence. We are interested in automated reasoning on large knowledge bases (KB) with rich and diverse semantics Suchanek et al. (2007); Bollacker et al. (2008); Carlson et al. (2010). KBs are highly incomplete Min et al. (2013), and facts not directly stored in a KB can often be inferred from those that are, creating exciting opportunities and challenges for automated reasoning. For example, consider the small knowledge graph in Figure 2.1. We can answer the question "Who did Malala Yousafzai share her Nobel Peace prize with?" from the following reasoning *path*: Malala Yousafzai $\rightarrow$ WonAward $\rightarrow$ Nobel Peace Prize 2014 $\rightarrow$ AwardedTo $\rightarrow$ Kailash Satyarthi. Our goal is to automatically learn such reasoning paths in KBs. We frame the learning problem as one of query answering, that is to say, answering questions of the form (Malala Yousafzai, SharesNobelPrizeWith, ?).

From its early days, the focus of automated reasoning approaches has been to build systems that can learn crisp symbolic logical rules McCarthy (1960); Nilsson (1991). Symbolic representations have also been integrated with machine learning especially in statistical relational learning Muggleton et al. (1992); Getoor and Taskar (2007); Kok and Domingos (2007); Lao et al. (2011), but due to poor generalization performance, these approaches have largely been superceded by distributed vector representations. Learning embedding of entities and relations using tensor factorization or neural methods has been a popular approach (Nickel et al., 2011a; Bordes et al., 2013; Socher et al., 2013, inter alia), but these methods cannot capture chains of reasoning expressed by KB paths. Neural multi-hop models Neelakantan et al. (2015a); Guu et al. (2015); Toutanova et al. (2016) address the aforementioned problems to some extent by operating on KB paths embedded in vector space. However, these models take as input a set of paths which are gathered by performing random walks *independent* of the query relation. Additionally, models such as those developed in Neelakantan et al. (2015a); Das et al. (2017) use the same set of initially collected paths to answer a diverse set of query types (e.g. MarriedTo, Nationality, WorksIn etc.).

This paper presents a method for efficiently searching the graph for answer-providing paths using reinforcement learning (RL) conditioned on the input question, eliminating any need for precomputed paths. Given a massive knowledge graph, we learn a policy, which, given the query $(entity_1, relation, ?)$, starts from $entity_1$ and learns to walk to the answer node by choosing to take a labeled relation edge at each step, *conditioning* on the query relation and entire path history. This formulates the query-answering task as a reinforcement learning (RL) problem where the goal is to take an optimal sequence of decisions (choices of relation edges) to maximize the expected reward (reaching the correct answer node). We call the RL agent MINERVA for "Meandering In Networks of Entities to Reach Verisimilar Answers."

Our RL-based formulation has many desirable properties. First, MINERVA has the built-in flexibility to take paths of variable length, which is important for answering harder questions that require complex chains of reasoning Shen et al. (2017). Secondly, MINERVA needs *no pretraining* and trains on the knowledge graph from scratch with reinforcement learning; no other supervision or fine-tuning is required representing a significant advance over prior applications of RL in NLP. Third, our path-based approach is computationally efficient, since by searching in a small neighborhood around the query entity it avoids ranking all entities in the KB as in prior work. Finally, the reasoning paths found by our agent automatically form an interpretable provenance for its predictions.

The main contributions of the paper are: (a) We present agent MINERVA, which learns to do query answering by walking on a knowledge graph conditioned on an input query, stopping when it reaches the answer node. The agent is trained using reinforcement learning, specifically policy gradients (§ 2.2). (b) We evaluate MINERVA on several benchmark datasets and compare favorably to Neural Theorem Provers (NTP) Rocktäschel and Riedel (2017a) and Neural LP Yang et al. (2017), which do logical rule learning in KBs, and also state-of-the-art embedding based methods such as DistMult Yang et al. (2015) and ComplEx Trouillon

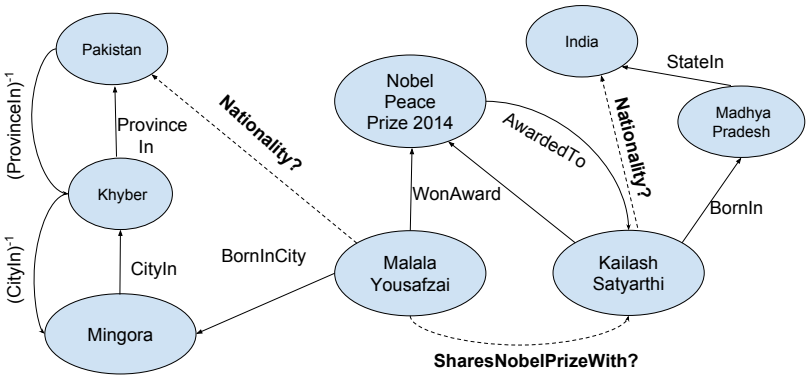


Figure 2.1: A small fragment of a knowledge base represented as a knowledge graph. Solid edges are observed and dashed edges are part of queries. Note how each query relation (e.g. SharesNobelPrizeWith, Nationality, etc.) can be answered by traversing the graph via "logical" paths between entity 'Malala Yousafzai' and the corresponding answer.

et al. (2016) and ConvE Dettmers et al. (2018a). (c) We also extend MINERVA to handle partially structured natural language queries and test it on the WikiMovies dataset (§ 2.3.3) Miller et al. (2016a).

We also compare to DeepPath Xiong et al. (2017) which uses reinforcement learning to pick paths between entity pairs. The main difference is that the state of their RL agent includes the answer entity since it is designed for the simpler task of predicting if a fact is true or not. As such their method cannot be applied directly to our more challenging query answering task where the second entity is unknown and must be inferred. Nevertheless, MINERVA outperforms DeepPath on their benchmark NELL-995 dataset when compared in their experimental setting (§ 2.3.2).

2.2 Task and Model

We formally define the task of query answering in a KB. Let $\mathcal{E}$ denote the set of entities and $\mathcal{R}$ denote the set of binary relations. A KB is a collection of facts stored as triplets $(\mathrm{e}_1, \mathrm{r}, \mathrm{e}_2)$ where $\mathrm{e}_1, \mathrm{e}_2 \in \mathcal{E}$ and $\mathrm{r} \in \mathcal{R}$. From the KB, a knowledge graph $\mathcal{G}$ can be constructed where the entities $\mathrm{e}_1, \mathrm{e}_2$ are represented as the nodes and relation r as labeled edge between them. Formally, a knowledge graph is a directed labeled multigraph $\mathcal{G} = (V, E, \mathcal{R})$, where V and E denote the vertices and edges of the graph respectively. Note that $V = \mathcal{E}$ and $E \subseteq V \times \mathcal{R} \times V$. Also, following previous approaches Bordes et al. (2013); Neelakantan et al. (2015a); Xiong et al. (2017), we add the inverse relation of every edge, i.e. for an edge $(\mathrm{e}_1, \mathrm{r}, \mathrm{e}_2) \in E$, we add the edge $(\mathrm{e}_2, \mathrm{r}^{-1}, \mathrm{e}_1)$ to the graph. (If the set of binary relations $\mathcal{R}$ does not contain the inverse relation r^{-1}, it is added to $\mathcal{R}$ as well.)

Since KBs have a lot of missing information, two natural tasks have emerged in the information extraction community - fact prediction and query answering. Query answering seeks to answer questions of the form $(\mathrm{e}_1, \mathrm{r}, ?)$, e.g. Toronto, locatedIn, ?, whereas fact prediction involves predicting if a fact is true or not, e.g. (Toronto, locatedIn, Canada)?. Algorithms for fact prediction can be used for query answering, but with significant computation overhead, since all candidate answer entities must be evaluated, making it prohibitively expensive for large KBs with millions of entities. In this work, we present a query answering model, that learns to efficiently traverse the knowledge graph to find the correct answer to a query, eliminating the need to evaluate all entities.

Query answering reduces naturally to a finite horizon sequential decision making problem as follows: We begin by representing the environment as a deterministic partially observed Markov decision process on a knowledge graph $\mathcal{G}$ derived from the KB (§2.2.1). Our RL agent is given an input query of the form $(\mathrm{e}_{1\mathrm{q}}, \mathrm{r}_\mathrm{q}, ?)$. Starting from vertex corresponding to e_{1q} in $\mathcal{G}$, the agent follows a path in the graph stopping at a node that it predicts as the answer (§ 2.2.2). Using a training set of known facts, we train the agent using policy gradients more specifically by REINFORCE Williams (1992) with control variates (§ 2.2.3). Let us begin by describing the environment.

2.2.1 Environment - States, Actions, Transitions and Rewards

Our environment is a finite horizon, deterministic partially observed Markov decision process that lies on the knowledge graph $\mathcal{G}$ derived from the KB. On this graph we will now specify a deterministic partially observed Markov decision process, which is a 5-tuple $(\mathcal{S}, \mathcal{O}, \mathcal{A}, \delta, R)$, each of which we elaborate below.

States. The state space $\mathcal{S}$ consists of all valid combinations in $\mathcal{E} \times \mathcal{E} \times \mathcal{R} \times \mathcal{E}$. Intuitively, we want a state to encode the query $(\mathrm{e_{1q}}, \mathrm{r_q})$, the answer $(\mathrm{e_{2q}})$, and a location of exploration $\mathrm{e_t}$ (current location of the RL agent). Thus overall a state $S \in \mathcal{S}$ is represented by $S = (\mathrm{e_t}, \mathrm{e_{1q}}, \mathrm{r_q}, \mathrm{e_{2q}})$ and the state space consists of all valid combinations.

Observations. The complete state of the environment is not observed. Intuitively, the agent knows its current location $(\mathrm{e_t})$ and $(\mathrm{e_{1q}}, \mathrm{r_q})$, but not the answer $(\mathrm{e_{2q}})$, which remains hidden. Formally, the observation function $\mathcal{O} : \mathcal{S} \to \mathcal{E} \times \mathcal{E} \times \mathcal{R}$ is defined as $\mathcal{O}(s = (\mathrm{e_t}, \mathrm{e_{1q}}, \mathrm{r_q}, \mathrm{e_{2q}})) = (\mathrm{e_t}, \mathrm{e_{1q}}, \mathrm{r_q})$.

Actions. The set of possible actions $\mathcal{A}_S$ from a state $S = (\mathrm{e_t}, \mathrm{e_{1q}}, \mathrm{r_q}, \mathrm{e_{2q}})$ consists of all outgoing edges of the vertex $\mathrm{e_t}$ in $\mathcal{G}$. Formally $\mathcal{A}_S = \{(\mathrm{e_t}, r, v) \in E : S = (\mathrm{e_t}, \mathrm{e_{1q}}, \mathrm{r_q}, \mathrm{e_{2q}}), r \in \mathcal{R}, v \in V\} \cup \{(s, \varnothing, s)\}$. Basically, this means an agent at each state has option to select which outgoing edge it wishes to take having the knowledge of the label of the edge r and destination vertex v.

During implementation, we unroll the computation graph up to a fixed number of time steps T. We augment each node with a special action called 'NO_OP' which goes from a node to itself. Some questions are easier to answer and needs fewer steps of reasoning than others. This design decision allows the agent to remain at a node for any number of time steps. This is especially helpful when the agent has managed to reach a correct answer at a time step $t < \mathrm{T}$ and can continue to stay at the 'answer node' for the rest of the time steps. Alternatively, we could have allowed the agent to take a special 'STOP' action, but we found the current setup to work sufficiently well. As mentioned before, we also add the inverse relation of a triple, i.e. for the triple (e_1, r, e_2), we add the triple (e_2, r^{-1}, e_1) to the graph. We found this important because this actually allows our agent to undo a potentially wrong decision.

Transition. The environment evolves deterministically by just updating the state to the new vertex incident to the edge selected by the agent. The query and answer remains the same. Formally, the transition function is $\delta : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ defined by $\delta(S, A) = (v, \mathrm{e_{1q}}, \mathrm{r_q}, \mathrm{e_{2q}})$, where $S = (\mathrm{e_t}, \mathrm{e_{1q}}, \mathrm{r_q}, \mathrm{e_{2q}})$ and $A = (\mathrm{e_t}, r, v))$.

Rewards. We only have a terminal reward of +1 if the current location is the correct answer at the end and 0 otherwise. To elaborate, if $S_T = (\mathrm{e_t}, \mathrm{e_{1q}}, \mathrm{r_q}, \mathrm{e_{2q}})$ is the final state, then we receive a reward of +1 if $\mathrm{e_t} = \mathrm{e_{2q}}$ else 0., i.e. $R(S_T) = \mathbb{I}\{\mathrm{e_t} = \mathrm{e_{2q}}\}$.

2.2.2 Policy Network

To solve the finite horizon deterministic partially observable Markov decision process described above, we design a randomized non-stationary history-dependent policy $\pi = (\mathbf{d_1}, \mathbf{d_2}, ..., \mathbf{d_{T-1}})$, where $\mathbf{d_t} : H_t \to \mathcal{P}(\mathcal{A}_{S_t})$ and history $H_t = (H_{t-1}, A_{t-1}, O_t)$ is just the sequence of observations and actions taken. We restrict ourselves to policies parameterized by long short-term memory network (LSTM) Hochreiter and Schmidhuber (1997).

An agent based on LSTM encodes the history H_t as a continuous vector $\mathbf{h_t} \in \mathbb{R}^{2d}$. We also have embedding matrix $\mathbf{r} \in \mathbb{R}^{|\mathcal{R}| \times d}$ and $\mathbf{e} \in \mathbb{R}^{|\mathcal{E}| \times d}$ for the binary relations and entities respectively. The history embedding for $H_t = (H_{t-1}, A_{t-1}, O_t)$ is updated according to LSTM dynamics:

$$\mathbf{h_t} = \mathrm{LSTM}\left(\mathbf{h_{t-1}}, [\mathbf{a_{t-1}}; \mathbf{o_t}]\right) \tag{2.1}$$

where $\mathbf{a_{t-1}} \in \mathbb{R}^d$ and $\mathbf{o_t} \in \mathbb{R}^d$ denote the vector representation for action/relation at time $t-1$ and observation/entity at time t respectively and $[;]$ denote vector concatenation. To elucidate, $\mathbf{a_{t-1}} = \mathbf{r}_{A_{t-1}}$, i.e. the embedding of the relation corresponding to label of the edge the agent chose at time $t-1$ and $\mathbf{o_t} = \mathbf{e}_{\mathrm{e}_t}$ if $O_t = (\mathrm{e_t}, \mathrm{e_{1q}}, \mathrm{r_q})$ i.e. the embedding of the entity corresponding to vertex the agent is at time t.

Based on the history embedding $\mathbf{h_t}$, the policy network makes the decision to choose an action from all available actions $(\mathcal{A}_{S_t})$ *conditioned* on the query relation. Recall that each possible action represents an

Dataset	#entities	#relations	#facts	#queries	#degree	
					avg.	median
COUNTRIES	272	2	1158	24	4.35	4
UMLS	135	49	5,216	661	38.63	28
KINSHIP	104	26	10686	1074	82.15	82
WN18RR	40,945	11	86,835	3134	2.19	2
NELL-995	75,492	200	154,213	3992	4.07	1
FB15K-237	14,505	237	272,115	20,466	19.74	14
WikiMovies	43,230	9	196,453	9952	6.65	4

Table 2.1: Statistics of various datasets used in experiments.

outgoing edge with information of the edge relation label l and destination vertex/entity d. So embedding for each $A \in \mathcal{A}_{S_t}$ is $[\mathbf{r_l}; \mathbf{e_d}]$, and stacking embeddings for all the outgoing edges we obtain the matrix $\mathbf{A_t}$. The network taking these as inputs is parameterized as a two-layer feed-forward network with ReLU nonlinearity which takes in the current history representation $\mathbf{h_t}$ and the embedding for the query relation $\mathbf{r_q}$ and outputs a probability distribution over the possible actions from which a discrete action is sampled. In other words,

$$\mathbf{d_t} = \text{softmax}\left(\mathbf{A_t}(\mathbf{W_2}\text{ReLU}\left(\mathbf{W_1}\left[\mathbf{h_t}; \mathbf{o_t}; \mathbf{r_q}\right]\right))\right),$$
$$A_t \sim \text{Categorical}\left(\mathbf{d_t}\right).$$

Note that the nodes in $\mathcal{G}$ do not have a fixed ordering or number of edges coming out from them. The size of matrix $\mathbf{A_t}$ is $|\mathcal{A}_{S_t}| \times 2d$, so the decision probabilities d_t lies on simplex of size $|\mathcal{A}_{S_t}|$. Also the procedure above is invariant to order in which edges are presented as desired and falls in purview of neural networks designed to be permutation invariant Zaheer et al. (2017). Finally, to summarize, the parameters of the LSTM, the weights $\mathbf{W_1}$, $\mathbf{W_2}$, the corresponding biases (not shown above for brevity), and the embedding matrices form the parameters θ of the policy network.

2.2.3 Training

For the policy network (π_θ) described above, we want to find parameters θ that maximize the expected reward:

$$J(\theta) = \mathbb{E}_{(e_1,r,e_2)\sim D}\mathbb{E}_{A_1,..,A_{T-1}\sim\pi_\theta}[R(S_T)|S_1 = (e_1, e_1, r, e_2)],$$

where we assume there is a true underlying distribution $(\mathrm{e}_1, \mathrm{r}, \mathrm{e}_2) \sim D$. To solve this optimization problem, we employ REINFORCE Williams (1992) as follows:

- The first expectation is replaced with empirical average over the training dataset.
- For the second expectation, we approximate by running multiple rollouts for each training example. The number of rollouts is fixed and for all our experiments we set this number to 20.
- For variance reduction, a common strategy is to use an additive control variate baseline Hammersley (2013); Fishman (2013); Evans and Swartz (2000). We use a moving average of the cumulative discounted reward as the baseline. We tune the weight of this moving average as a hyperparameter. Note that in our experiments we found that using a learned baseline performed similarly, but we finally settled for cumulative discounted reward as the baseline owing to its simplicity.
- To encourage diversity in the paths sampled by the policy at training time, we add an entropy regularization term to our cost function scaled by a constant (β).

2.3 Experiments

We now present empirical studies for MINERVA in order to establish that (i) MINERVA is competitive for query answering on small (Sec. 2.3.1) as well as large KBs (Sec. 2.3.1), (ii) MINERVA is superior to a path based

	ComplEx	ConvE	DistMult	NTP	NTP-λ	NeuralLP	MINERVA
S1	99.37±0.4	**100.0±0.00**	97.91±0.01	90.83±15.4	**100.0±0.00**	**100.0±0.0**	**100.0±0.00**
S2	87.95±2.8	**99.0±1.00**	69.18±2.38	87.40±11.7	93.04±0.40	75.1 ± 0.3	92.36±2.41
S3	48.44±6.3	86.0±5.00	15.79±0.64	56.68±17.6	77.26±17.0	92.2 ± 0.2	**95.10±1.20**

Table 2.2: Performance on three tasks of COUNTRIES dataset with AUC-PR metric. MINERVA significantly outperforms all other methods on the hardest task (S3). Also variance across runs for MINERVA is lower compared to other methods.

models that do not search the KB efficiently or train query specific models (Sec. 2.3.2), (iii) MINERVA can not only be used for well formed queries, but can also easily handle partially structured natural language queries (Sec 2.3.3), (iv) MINERVA is highly capable of reasoning over long chains, and (v) MINERVA is robust to train and has much faster inference time (Sec. 2.3.5).

2.3.1 Knowledge Base Query Answering

To gauge the reasoning capability of MINERVA, we begin with task of query answering on KB, i.e. we want to answer queries of the form $(e_1, r, ?)$. Note that, as mentioned in Sec. 2.2, this task is subtly different from fact checking in a KB. Also, as most of the previous literature works in the regime of fact checking, their ranking includes variations of both (e_1, r, x) and (x, r, e_2). However, since we do not have access to e_2 in case of question answering scenario the same ranking procedure does not hold for us – we only need to rank on (e_1, r, x). This difference in ranking made it necessary for us to re-run all the implementations of previous work. We used the implementation or the best pre-trained models (whenever available) of Rocktäschel and Riedel (2017a); Yang et al. (2017) and Dettmers et al. (2018a). For MINERVA to produce a ranking of answer entities during inference, we do a beam search with a beam width of 50 and rank entities by the probability of the trajectory the model took to reach the entity and remaining entities are given a rank of ∞.

Method We compare MINERVA with various state-of-the-art models using HITS@1,3,10 and mean reciprocal rank (MRR), which are standard metrics for KB completion tasks. In particular we compare against embedding based models - DistMult Yang et al. (2015), ComplEx Trouillon et al. (2016) and ConvE Dettmers et al. (2018a). For ConvE and ComplEx, we used the implementation released by Dettmers et al. (2018a)[1] on the best hyperparameter settings reported by them. For DistMult, we use our highly tuned implementation (e.g. which performs better than the state-of-the-art results of Toutanova et al. (2015)). We also compare with two recent work in learning logical rules in KB namely Neural Theorem Provers (NTP) Rocktäschel and Riedel (2017a) and NeuralLP Yang et al. (2017). Rocktäschel and Riedel (2017a) also reports a NTP model which is trained with an additional objective function of ComplEx (NTP-λ). For these models, we used the implementation released by corresponding authors [2] [3], again on the best hyperparameter settings reported by them.

Smaller Datasets

Dataset We use three standard datasets: COUNTRIES Bouchard et al. (2015), KINSHIP, and UMLS Kok and Domingos (2007). The COUNTRIES dataset ontains countries, regions, and subregions as entities and is carefully designed to explicitly test the logical rule learning and reasoning capabilities of link prediction models. The queries are of the form LocatedIn(c, ?) and the answer is a region (e.g. LocatedIn(Egypt, ?) with the answer as Africa). The dataset has 3 tasks (S1-3 in table 2.2) each requiring reasoning steps of increasing length and difficulty (see Rocktäschel and Riedel (2017a) for more details about the tasks). Following the design of the COUNTRIES dataset, for task S1 and S2, we set the maximum path length $T = 2$ and for S3, we set $T = 3$. The Unified Medical Language System (UMLS) dataset, is from biomedicine. The entities are biomedical concepts (e.g. disease, antibiotic) and relations are like treats and diagnoses. The KINSHIP

[1] https://github.com/TimDettmers/ConvE
[2] https://github.com/uclmr/ntp
[3] https://github.com/fanyangxyz/Neural-LP

Data	Metric	ComplEx	ConvE	DistMult	NTP	NTP-λ	NeuralLP	MINERVA
KINSHIP	HITS@1	0.754	0.697	0.808	0.500	0.759	0.475	0.605
	HITS@3	0.910	0.886	0.942	0.700	0.798	0.707	0.812
	HITS@10	0.980	0.974	0.979	0.777	0.878	0.912	0.924
	MRR	0.838	0.797	0.878	0.612	0.793	0.619	0.720
UMLS	HITS@1	0.823	0.894	0.916	0.817	0.843	0.643	0.728
	HITS@3	0.962	0.964	0.967	0.906	0.983	0.869	0.900
	HITS@10	0.995	0.992	0.992	0.970	1.000	0.962	0.968
	MRR	0.894	0.933	0.944	0.872	0.912	0.778	0.825

Table 2.3: Query answering results on KINSHIP and UMLS datasets.

dataset contains kinship relationships among members of the Alyawarra tribe from Central Australia. For these two task we use maximum path length $T = 2$. Also, for MINERVA we turn off entity in (2.1) in these experiments.

Observations For the COUNTRIES dataset, in Table 2.2 we report a stronger metric - the area under the precision-recall curve - as is common in the literature. We can see that MINERVA compares favorably or outperforms all the baseline models except on the task S2 of COUNTRIES, where the ensemble model NTP-λ and ConvE outperforms it, albeit with a higher variance across runs. Our gains are much more prominent in task S3, which is the hardest among all the tasks.

The Kinship and UMLS datasets are small KB datasets with around 100 entities each and as we see from Table 2.3, embedding based methods (ConvE, ComplEx and DistMult) perform much better than methods which aim to learn logical rules (NTP, NeuralLP and MINERVA). On Kinship, MINERVA outperforms both NeuralLP and NTP and matches the HITS@10 performance of NTP on UMLS. Unlike COUNTRIES, these datasets were not designed to test the logical rule learning ability of models and given the small size, embedding based models are able to get really high performance. Combination of both methods gives a slight increase in performance as can be seen from the results of NTP-λ. However, when we initialized MINERVA with pre-trained embeddings of ComplEx, we did not find a significant increase in performance.

Larger Datasets

Dataset Next we evaluate MINERVA on three large KG datasets - WN18RR, FB15K-237 and NELL-995. The WN18RR Dettmers et al. (2018a) and FB15K-237 Toutanova et al. (2015) datasets are created from the original WN18 and FB15K datasets respectively by removing various sources of test leakage, making the datasets more realistic and challenging. The NELL-995 dataset released by Xiong et al. (2017) has separate graphs for each query relation, where a graph for a query relation can have triples from the test set of another query relation. For the query answering experiment, we combine all the graphs and removed all test triples (and the corresponding triples with inverse relations) from the graph. We also noticed that several triples in the test set had an entity (source or target) that never appeared in the graph. Since, there will be no trained embeddings for those entities, we removed them from the test set. This reduced the size of test set from 3992 queries to 2818 queries.[4]

Observations Table 2.4 reports the query answering results on the larger WN18RR, FB15K-237 and NELL-995 datasets. We could not include the results of NeuralLP on NELL-995 since it didn't scale to that size. Similarly NTP did not scale to any of the larger datasets. Apart from these, we are the first to report a comprehensive summary of performance of all baseline methods on these datasets.

On NELL-995, MINERVA performs comparably to embedding based methods such as DistMult and ComplEx and performs comparably with ConvE on the stricter HITS@1 metric. ConvE, however outperforms us on HITS@10 on NELL-995. On WN18RR, logic based based methods (NeuralLP, MINERVA) generally outperform

[4] Available at `https://github.com/shehzaadzd/MINERVA`

Data	Metric	ComplEx	ConvE	DistMult	NeuralLP	Path-Baseline	MINERVA
WN18RR	HITS@1	0.382	0.403	0.410	0.376	0.017	0.413
	HITS@3	0.433	0.452	0.441	0.468	0.025	0.456
	HITS@10	0.480	0.519	0.475	0.657	0.046	0.513
	MRR	0.415	0.438	0.433	0.463	0.027	0.448
FB15K-237	HITS@1	0.303	0.313	0.275	0.166	0.169	0.217
	HITS@3	0.434	0.457	0.417	0.248	0.248	0.329
	HITS@10	0.572	0.600	0.568	0.348	0.357	0.456
	MRR	0.394	0.410	0.370	0.227	0.227	0.293
NELL-995	HITS@1	0.612	0.672	0.610	-	0.300	0.663
	HITS@3	0.761	0.808	0.733	-	0.417	0.773
	HITS@10	0.827	0.864	0.795	-	0.497	0.831
	MRR	0.694	0.747	0.680	-	0.371	0.725

Table 2.4: Query answering results on WN18RR, FB15K-237 and NELL-995 datasets. NeuralLP does not scale to NELL-995 and hence the entries are kept blank.

embedding based methods, with MINERVA achieving the highest score on HITS@1 metric and NeuralLP significantly outperforming on HITS@10.

We observe that on FB15K-237, however, embedding based methods dominate over MINERVA and NeuralLP. Upon deeper inspection, we found that the query relation types of FB15K-237 knowledge graph differs significantly from others.

Analysis of query relations of FB15k-237: We analyzed the type of query relation types on the FB15K-237 dataset. Following Bordes et al. (2013), we categorized the query relations into (M)any to 1, 1 to M or 1 to 1 relations. An example of a M to 1 relation would be '/people/profession' (What is the profession of person 'X'?). An example of 1 to M relation would be /music/instrument/instrumentalists ('Who plays the music instrument X?') or '/people/ethnicity/people' ('Who are people with ethnicity X?'). From a query answering point of view, the answer to these questions is a list of entities. However, during evaluation time, the model is evaluated based on whether it is able to predict the one target entity which is in the query triple. Also, since MINERVA outputs the end points of the paths as target entities, it is sometimes possible that the particular target entity of the triple does not have a path from the source entity (however there are paths to other 'correct' answer entities). Table 2.5 shows few other examples of relations belonging to different classes.

Following Bordes et al. (2013), we classify a relation as 1-to-M if the ratio of cardinality of tail to head entities is greater than 1.5 and as M-to-1 if it is lesser than 0.67. In the validation set of FB15K-237, 54% of the queries are 1-to-M, whereas only 26% are M-to-1. Contrasting it with NELL-995, 27% are 1-to-M and 36% are M-to-1 or UMLS where only 18% are 1-to-M. Table 2.6 shows few relations from FB15K-237 dataset which have high tail-to-head ratio. The average ratio for 1-TO-M relations in FB15K-237 is **13.39** (substantially higher than 1.5). As explained before, the current evaluation scheme is not suited when it comes to 1-to-M relations and the high percentage of 1-to-M relations in FB15K-237 also explains the sub optimal performance of MINERVA.

We also check the frequency of occurrence of various unique path types. We define a path type as the sequence of relation types (ignoring the entities) in a path. Intuitively, a predictive path which generalizes across queries will occur many number of times in the graph. Figure 2.2 shows the plot. As we can see, the characteristics of FB15K-237 is quite different from other datasets. For example, in NELL-995, more than 1000 different path types occur more than 1000 times. WN18RR has only 11 different relation types which means there are only 11^3 possible path types of length 3 and even fewer number of them would be predictive. As can be seen, there are few path types which occur more than 10^4 times and around 50 of them occur more than 1000 times. However in FB15K-237, which has the highest number of relation types, we observe a sharp decrease in the number of path types which occur a significant number of times. Since MINERVA cannot find path types which repeat often, it finds it hard to learn path types that generalize.

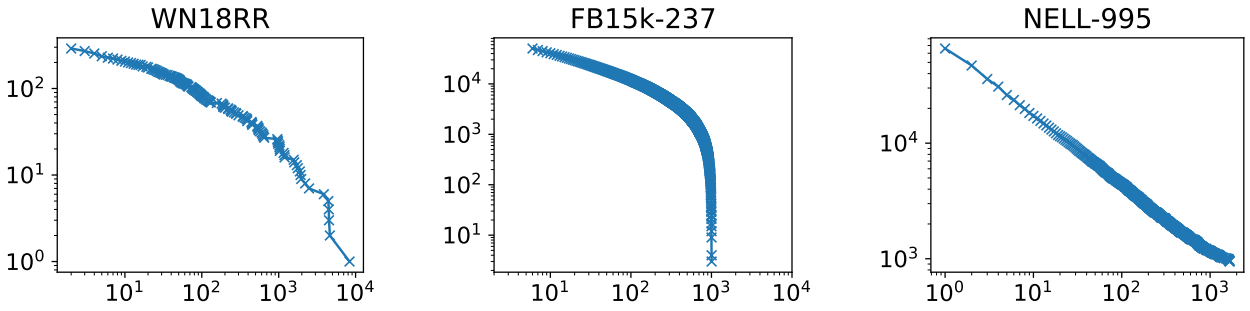


Figure 2.2: Count of number of unique path types of length 3 which occur more than 'x' times in various datasets. For example, in NELL-995 there are more than 10^3 path types which occur more than 10^3 times. However, for FB15k-237, we see a sharp decrease as 'x' becomes higher, suggesting that path types do not repeat often.

(i) **M to 1**
Los Angeles Rams $\xrightarrow{\text{team plays sport}}$ American Football
The Walking Dead $\xrightarrow{\text{country of origin}}$ USA
(ii) **1 to M**
CEO $\xrightarrow{\text{job position in organization}}$ Merck & Co.
Traffic collision $\xrightarrow{\text{cause of death}}$ Albert Camus
Harmonica $\xrightarrow{\text{instrument played by musician}}$ Greg Graffin

Table 2.5: Few example facts belonging to m to 1, 1 to m relations in FB15K-237

.

2.3.2 Comparison with Path based models

With Random Walk Models

In this experiment, we compare to a model which gathers path based on random walks and tries to predict the answer entity. Neural multi-hop models Neelakantan et al. (2015a); Toutanova et al. (2016), operate on paths between entity pairs in a KB. However these methods need to know the target entity in order to pre-compute paths between entity pairs. Guu et al. (2015) is an exception in this regard as they do random walks starting from a source entity 'e_1' and then using the path, they train a classifier to predict the target answer entity. However, they only consider *one* path starting from a source entity. In contrast, Neelakantan et al. (2015a); Toutanova et al. (2016) use information from multiple paths between the source and target entity. We design a baseline model which combines the strength of both these approaches. Starting from 'e_1' , the model samples $(k = 100)$ random paths of up to a maximum length of $T = 3$. Following Neelakantan et al. (2015a), we encode each paths with an LSTM followed by a max-pooling operation to featurize the paths. This feature is concatenated with the source entity and query relation vector which is then passed through a feed forward network which scores all possible target entities. The network is trained with a multi-class cross entropy objective based on observed triples and during inference we rank target entities according to the model score.

The PATH-BASELINE column of table 2.4 shows the performance of this model on the three datasets. As we can see MINERVA outperforms this baseline significantly. This shows that a model which predicts based on a set of randomly sampled paths does not do as well as MINERVA because it either loses important paths during random walking or it fails to aggregate predictive features from all the k paths, many of which would be irrelevant to answer the given query. The latter is akin to the problem with distant supervision Mintz et al. (2009), where important evidence gets lost amidst a plethora of irrelevant information. However, by taking each step conditioned on the query relation, MINERVA can effectively reduce the search space and focus on paths relevant to answer the query.

Relation	tail/head
/people/marriage_union_type/unions_of_this_type./people/marriage/location_of_ceremony	129.75
/organization/role/leaders./organization/leadership/organization	65.15
/location/country/second_level_divisions	49.18
/user/ktrueman/default_domain/international_organization/member_states	36.5
/base/marchmadness/ncaa_basketball_tournament/seeds./base/marchmadness/ncaa_tournament_seed/team	33.6

Table 2.6: Few example 1-to-M relations from FB15K-237 with high cardinality ratio of tail to head.

Task	**DeepPath**	**MINERVA**	**MINERVA**[a]
athleteplaysinleague	0.960	**0.970**	0.940
worksfor	0.711	**0.825**	**0.810**
organizationhiredperson	0.742	**0.851**	**0.856**
athleteplayssport	0.957	**0.985**	**0.980**
teamplayssport	0.738	**0.846**	**0.880**
personborninlocation	0.757	**0.793**	**0.780**
personleadsorganization	0.795	**0.851**	**0.877**
athletehomestadium	0.890	**0.895**	**0.898**
organizationheadquarteredincity	0.790	**0.946**	**0.940**
athleteplaysforteam	0.750	**0.824**	**0.800**

Table 2.7: MAP scores for different query relations on the NELL-995 dataset. Note that in this comparison, MINERVA refers to only a single learnt model for all query relations which is competitive with individual DeepPath models trained separately for each query relation. We also trained MINERVA in the setting of DeepPath, i.e. training per-relation models (MINERVA[a])

With DeepPath

We also compare MINERVA with DeepPath which uses RL to pick paths between entity pairs. For a fair comparison, we only rank the answer entities against the negative examples in the dataset used in their experiments[5] and report the mean average precision (MAP) scores for each query relation. DeepPath feeds the paths its agent gathers as input features to the path ranking algorithm (PRA) Lao et al. (2011), which trains a per-relation classifier. But unlike them, we train one model which learns for all query relations so as to enable our agent to leverage from correlations and more data. If our agent is not able to reach the correct entity or one of the negative entities, the corresponding entities gets a score of negative infinity. If MINERVA fails to reach any of the entities in the set of correct and negative entities. then we fall back to a random ordering of the entities. As show in table 2.7, we outperform them or achieve comparable performance for all the query relations For this experiment, we set the maximum length $T = 3$. Although training per-relation models is cumbersome and does not scale to massive KBs with thousands of relation types, we also train per-relation models of MINERVA replicating the settings of DeepPath (MINERVA[a] in table 2.7). MINERVA[a] outperforms DeepPath and performs similarly to MINERVA which is an encouraging result since training one model which performs well for all relation is highly desirable.

2.3.3 Partially Structured Queries

Queries in KBs are structured in the form of triples. However, this is unsatisfactory since for most real applications, the queries appear in natural language. As a first step in this direction, we extend MINERVA to take in "partially structured" queries. We use the WikiMovies dataset Miller et al. (2016a) which contains questions in natural language albeit generated by templates created by human annotators. An example question is "Which is a film written by Herb Freed?". WikiMovies also has an accompanying KB which can be used to answer all the questions.

Model	Accuracy
Memory Network	78.5
QA system	93.5
Key-Value Memory Network	93.9
Neural LP	94.6
MINERVA	**96.7**

Table 2.8: Performance on WikiMovies

[5]We are grateful to Xiong et al. (2017) for releasing the negative examples used in their experiments.

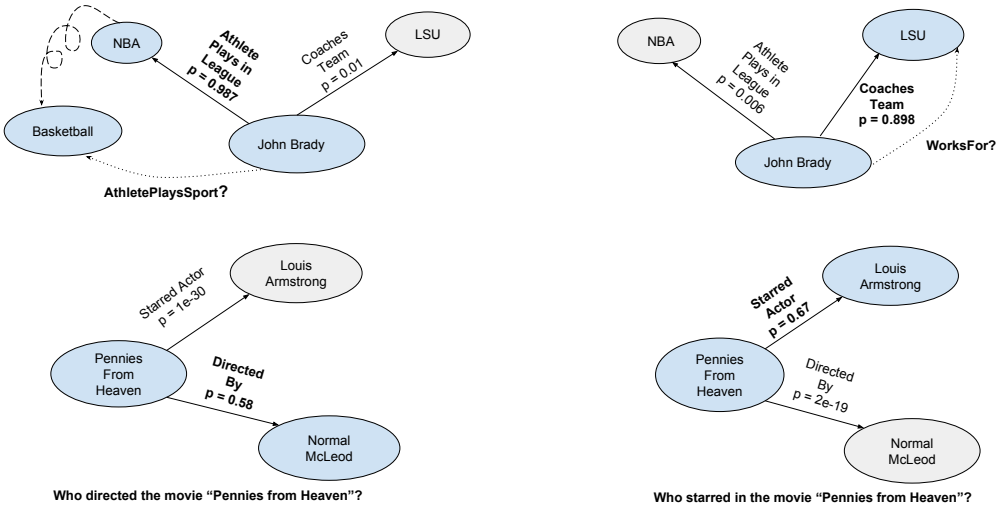


Figure 2.4: Based on the query relation our agent assigns different probabilities to different actions. The dashed edges in the top row denote query relation. Examples in the bottom row are from the WikiMovies dataset and hence the ques

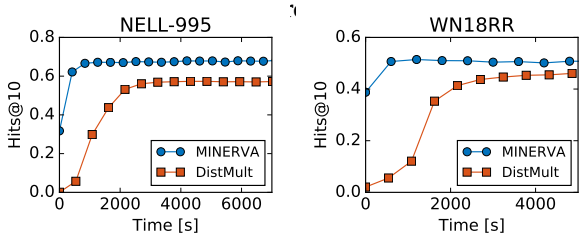


Figure 2.5: HITS@10 on the development set versus training time.

We link the entity occurring in the question to the KB via simple string matching. To form the vector representation of the query relation, we design a simple question encoder which computes the average of the embeddings of the question words. The word embeddings are learned from scratch and we do not use any pretrained embeddings. We compare our results with those reported in Yang et al. (2017) (table 2.8). For this experiment, we found that $T = 1$ sufficed, suggesting that WikiMovies is not the best testbed for multihop reasoning, but this experiment is a promising first step towards the realistic setup of using KBs to answer natural language question.

2.3.4 Grid World Path Finding

While chains in KB need not be very long to get good empirical results Neelakantan et al. (2015a); Das et al. (2017); Yang et al. (2017), in principle MINERVA can be used to learn long reasoning chains. To evaluate the same, we test our model on a synthetic 16-by-16 grid world dataset created by Yang et al. (2017), where the task is to navigate to a particular cell (answer entity) starting from a random cell (start entity) by following a set of directions (query relation). The KB consists of atomic triples of the form ((2,1), North, (1,1)) – entity (1,1) is north of entity (2,1). The queries consists of a sequence of directions (e.g. North, SouthWest, East). The queries are classified into classes based on the path lengths. Figure 2.3 shows the accuracy on varying path lengths. Compared to Neural LP, MINERVA is much more robust to queries, which require longer path, showing minimal degradation in performance for even the longest path in the dataset.

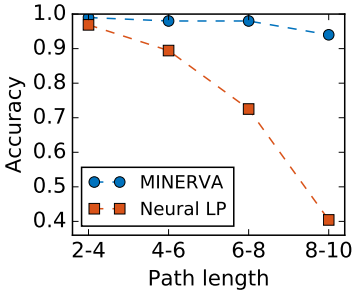


Figure 2.3: Grid world experiment: We significantly outperform NeuralLP for longer path lengths.

2.3.5 Further Analysis

Training time. Figure 2.5 plots the HITS@10 scores on the development set against the training time comparing MINERVA with DistMult. It can be seen that MINERVA converges to a higher score much faster than DistMult. It is also interesting to note that even during the early stages of the training, MINERVA has much higher performance than that of DistMult, as during these initial stages, MINERVA would just be doing random walks in the neighborhood of the source entity (e_1). This implies that MINERVA's approach of searching for an answer in the neighborhood of e_1 is a much more efficient and smarter strategy than ranking all entities in the knowledge graph (as done by DistMult and other related methods).

Inference Time. At test time, embedding based methods such as ConvE, ComplEx and DistMult rank all entities in the graph. Hence, for a test-time query, the running time is always $\mathcal{O}(|\mathcal{E}|)$ where $\mathcal{R}$ denotes the set of entities (= nodes) in the graph. MINERVA, on the other hand is efficient at inference time since it has to essentially search for answer entities in its local neighborhood. The many cost at inference time for MINERVA is to compute probabilities for all outgoing edges along the path. Thus inference time of MINERVA only depends on degree distribution of the graph. If we assume the knowledge graph to obey a power law degree distribution, like many natural graphs, then for MINERVA the average inference time can be shown to be $O(\frac{\alpha}{\alpha-1})$, when the coefficient of the power law $\alpha > 1$. The median inference time for MINERVA is $O(1)$ for all values of α. Note that these quantities are independent of size of entities $|\mathcal{E}|$. For instance, on the test dataset of WN18RR, the wall clock inference time of MINERVA is **63s** whereas that of a GPU implementation of DistMult, which is the simplest among the lot, is **211s**. Similarly the wall-clock inference time on the test set of NELL-995 for a GPU implementation of DistMult is 115s whereas that of MINERVA is **35s**.

Query based Decision Making. At each step before making a decision, our agent conditions on the query relation. Figure 2.4 shows examples, where based on the query relation, the probabilities are peaked on different actions. For example, when the query relation is WorksFor, MINERVA assigns a much higher probability of taking the edge CoachesTeam than AthletePlaysInLeague. We also see similar behavior on the WikiMovies dataset where the query consists of words instead of fixed schema relation.

Model Robustness. Table 2.9 also reports the mean and standard deviation across three independent runs of MINERVA. We found it easy to obtain/reproduce the highest scores across several runs as can be seen from the low deviations in scores.

Effectiveness of Remembering Path History. MINERVA encodes the history of decisions it has taken in the past using LSTMs. To test the importance of remembering the sequence of decisions, we did an ablation study in which the agent chose the next action based on only local information i.e. current entity and query and did not have access to the history $\mathrm{h_t}$. For the KINSHIP dataset, we observe a 27% points decrease in HITS@1 and 13% decrease in HITS@10. For grid-world, it is also not surprising that we see a big drop in performance. The final accuracy is 0.23 for path lengths 2-4 and 0.04 for lengths 8-10. For FB15K-237 the HITS@10 performance dropped from 0.456 to 0.408.

NO-OP and Inverse Relations. At each step, MINERVA can choose to take a NO-OP edge and remain at the same node. This gives the agent the flexibility of taking paths of variable lengths. Some questions are easier to answer than others and require fewer steps of reasoning and if the agent reaches the answer early, it can choose to remain there. Example (i) in table 2.10 shows such an example. Similarly inverse relation gives the agent the ability to recover from a potentially wrong decision it has taken before. Example (ii) shows such an example, where the agent took a incorrect decision at the first step but was able to revert the decision because of the presence of inverted edges.

2.4 Related Work

Learning vector representations of entities and relations using tensor factorization Nickel et al. (2011a, 2012); Bordes et al. (2013); Riedel et al. (2013); Nickel et al. (2014); Yang et al. (2015) or neural methods Socher

Dataset	HITS@1	HITS@3	HITS@10
NELL-995	0.66 ± 0.029	0.77 ± 0.0016	0.83 ± 0.0030
FB15K-237	0.22 ± 0.002	0.33 ± 0.0008	0.46 ± 0.0006
WN18RR	0.41 ± 0.030	0.45 ± 0.0180	0.51 ± 0.0005

Table 2.9: Mean and Standard deviation across runs for various datasets.

et al. (2013); Toutanova et al. (2015); Verga et al. (2016a) has been a popular approach to reasoning with a knowledge base. However, these methods cannot capture more complex reasoning patterns such as those found by following inference paths in KBs. Multi-hop link prediction approaches Lao et al. (2011); Neelakantan et al. (2015a); Guu et al. (2015); Toutanova et al. (2016); Das et al. (2017) address the problems above, but the reasoning paths that they operate on are gathered by performing random walks independent of the type of query relation. Lao et al. (2011) further filters paths from the set of sampled paths based on the restriction that the path must end at one of the target entities in the training set and are within a maximum length. These constraints make them query dependent but they are heuristic in nature. Our approach eliminates any necessity to pre-compute paths and learns to efficiently search the graph conditioned on the input query relation.

Inductive Logic Programming (ILP) Muggleton et al. (1992) aims to learn general purpose predicate rules from examples and background knowledge. Early work in ILP such as FOIL Quinlan (1990), PROGOL Muggleton (1995) are either rule-based or require negative examples which is often hard to find in KBs (by design, KBs store true facts). Statistical relational learning methods Getoor and Taskar (2007); Kok and Domingos (2007); Schoenmackers et al. (2010) along with probabilistic logic Richardson and Domingos (2006); Broecheler et al. (2010); Wang et al. (2013) combine machine learning and logic but these approaches operate on symbols rather than vectors and hence do not enjoy the generalization properties of embedding based approaches.

There are few prior work which treat inference as search over the space of natural language. Nogueira and Cho (2016) propose a task (WikiNav) in which each the nodes in the graph are Wikipedia pages and the edges are hyperlinks to other wiki pages. The entity is to be represented by the text in the page and hence the agent is required to reason over natural language space to navigate through the graph. Similar to WikiNav is Wikispeedia West et al. (2009) in which an agent needs to learn to traverse to a given target entity node (wiki

(i) **Can learn general rules:**

(S1) $\text{LocatedIn}(X, Y) \leftarrow \text{LocatedIn}(X, Z)$ & $\text{LocatedIn}(Z, Y)$
(S2) $\text{LocatedIn}(X, Y) \leftarrow \text{NeighborOf}(X, Z)$ & $\text{LocatedIn}(Z, Y)$
(S3) $\text{LocatedIn}(X, Y) \leftarrow \text{NeighborOf}(X, Z)$ & $\text{NeighborOf}(Z, W)$ & $\text{LocatedIn}(W, Y)$

(ii) **Can learn shorter path:** Richard F. Velky $\xrightarrow{\text{WorksFor}}$?

Richard F. Velky $\xrightarrow{\text{PersonLeadsOrg}}$ Schaghticokes $\xrightarrow{\text{NO-OP}}$ Schaghticokes $\xrightarrow{\text{NO-OP}}$ Schaghticokes

(iii) **Can recover from mistakes:** Donald Graham $\xrightarrow{\text{WorksFor}}$?

Donald Graham $\xrightarrow{\text{OrgTerminatedPerson}}$ TNT Post $\xrightarrow{\text{OrgTerminatedPerson}^{-1}}$ Donald Graham $\xrightarrow{\text{OrgHiredPerson}}$ Wash Post

Table 2.10: A few example of paths found by MINERVA on the COUNTRIES and NELL. MINERVA can learn general rules as required by the COUNTRIES dataset (example (i)). It can learn shorter paths if necessary (example (ii)) and has the ability to correct a previously taken decision (example (iii))

.

page) as quickly as possible. Angeli and Manning (2014) propose natural logic inference in which they cast the inference as a search from a query to any valid premise. At each step, the actions are one of the seven lexical relations introduced by MacCartney and Manning (2007).

Neural Theorem Provers (NTP) Rocktäschel and Riedel (2017a) and Neural LP Yang et al. (2017) are methods to learn logical rules that can be trained end-to-end with gradient based learning. NTPs are constructed by Prolog's backward chaining inference method. It operates on vectors rather than symbols, thereby providing a success score for each proof path. However, since a score can be computed between any two vectors, the computation graph becomes quite large because of such *soft-matching* during substitution step of backward chaining. For tractability, it resorts to heuristics such as only keeping the top-K scoring proof paths trading-off guarantees for exact gradients. Also the efficacy of NTPs has yet to be shown on large KBs. Neural LP introduces a differential rule learning system using operators defined in TensorLog Cohen (2016). It has a LSTM based controller with a differentiable memory component Graves et al. (2014); Sukhbaatar et al. (2015) and the rule scores are calculated via attention. Even though, differentiable memory allows end to end training, it necessitates accessing the entire memory, which can be computationally expensive. RL approaches capable of hard selection of memory Zaremba and Sutskever (2015) are computationally attractive. MINERVA uses a similar hard selection of relation edges to walk on the graph. More importantly, MINERVA outperforms both these methods on their respective benchmark datasets.

DeepPath Xiong et al. (2017) uses RL based approaches to find paths in KBs. However, the state of their MDP requires the target entity to be known in advance and hence their path finding strategy is dependent on knowing the answer entity. MINERVA does not need any knowledge of the target entity and instead learns to find the answer entity among all entities. DeepPath, additionally feeds its gathered paths to Path Ranking Algorithm Lao et al. (2011), whereas MINERVA is a complete system trained to do query answering. DeepPath also uses fixed pretrained embeddings for its entity and relations. Lastly, on comparing MINERVA with DeepPath in their experimental setting on the NELL dataset, we match their performance or outperform them. MINERVA is also similar to methods for learning to search for structured prediction Collins and Roark (2004); Daumé III and Marcu (2005); Daumé III et al. (2009); Ross et al. (2011); Chang et al. (2015). These methods are based on imitating a reference policy (oracle) which make near-optimal decision at every step. In our problem setting, it is unclear what a good reference policy would be. For example, a shortest path oracle between two entities would be unideal, since the answer providing path should depend on the query relation.

2.5 Conclusion

We explored a new way of automated reasoning on large knowledge bases in which we use the knowledge graphs representation of the knowledge base and train an agent to walk to the answer node conditioned on the input query. We achieve state-of-the-art results on multiple benchmark knowledge base completion tasks and we also show that our model is robust and can learn long chains-of-reasoning. Moreover it needs no pretraining or initial supervision. Future research directions include applying more sophisticated RL techniques and working directly on textual queries and documents.

3 Non-Parametric Reasoning on Knowledge Bases

3.1 Introduction

Given a new problem, humans possess the innate ability to 'retrieve' and 'adapt' solutions to *similar* problems from the past. For example, an automobile mechanic might fix a car engine by recalling previous experiences where cars exhibited similar symptoms of damage. This model of reasoning has been widely studied and verified in cognitive psychology for various applications such as mathematical problem solving Ross (1984), diagnosis by physicians Schmidt et al. (1990), automobile mechanics Lancaster and Kolodner (1987) etc. A lot of classical work in artificial intelligence (AI), particularly in the field of *case-based reasoning* (CBR) has focused on incorporating such kind of reasoning in AI systems (Schank, 1982; Kolodner, 1983; Rissland, 1983; Aamodt and Plaza, 1994; Leake, 1996, inter-alia).

At a high level, a case-based reasoning system is comprised of four steps Aamodt and Plaza (1994) — (a) 'retrieve', in which given a new problem, 'cases' that are similar to the given problem are retrieved. A 'case' is usually associated with a problem description (used for matching it to a new problem) and its corresponding solution. After the initial retrieval step, the previous solutions are (b) 'reused' for the problem in hand. Often times, however, the retrieved solutions cannot be directly used, and hence the solutions needs to be (c) 'revised'. Lastly, if the revised solution is useful for solving the given problem, they are (d) 'retained' in a memory so that they can be used in the future.

Knowledge graphs (KGs) Suchanek et al. (2007); Bollacker et al. (2008); Carlson et al. (2010) contain rich facts about entities and capture relations with diverse semantics between those entities. However, KGs can be highly incomplete Min et al. (2013), missing important edges (relations) between entities. For example, consider the small fragment of the KG around the entity MELINDA GATES in figure 3.1. Even though a lot of facts about MELINDA is captured, it is missing the edge corresponding to works_in_city. A recent series of work address this problem by modeling multi-hop paths between entities in the KG, thereby able to reason along the path MELINDA $\rightarrow$ ceo $\rightarrow$ GATES FOUNDATION $\rightarrow$ headquartered $\rightarrow$ SEATTLE. However, the number of paths starting from an entity increases exponentially w.r.t the path length and therefore past work used parametric models to do approximate search using reinforcement learning (RL) Xiong et al. (2017); Das et al. (2018); Lin et al. (2018a). Using RL-based methods have their own shortcomings, like hard to train and high computational requirements. Moreover, these models try to encode all the rules for reasoning into the parameters of the model which makes learning even harder.

In this paper, we propose a simple non-parametric approach for reasoning in KGs (Figure 3.1). Given an entity and a query relation (e_q, r_q), we first retrieve k entities in the KG that are similar to e_q and for which we observe the query relation edge r_q. The retrieved entities could be present anywhere in the KG and are not just restricted in the immediate neighborhood of e_q. Similarity between entities is measured based on the observed relations that the entities participate in. This ensures that the retrieved entities have similar observed properties as e_q (e.g., if e_q is a CEO, then the retrieved entities are also CEO's or business person). Next, for each of the retrieved entities, our method finds a set of reasoning paths that connect the retrieved entities to the entities that they are connected with via the query relation r_q. In this way, our method removes the burden of storing the reasoning rules in the parameters of the model and rather extract it from entities similar to the query entity. Next, our method checks if similar reasoning paths exists starting from the query entity e_q. If similar paths exists, then the answer to the original query is found by starting from e_q and traversing the KG by following the reasoning path. In practice, we find *multiple* reasoning paths which end at different entities and we rank the entities based on the number of reasoning paths that lead to them, with the intuition that an entity supported by multiple reasoning paths is likely to be a better answer to the given query.

Apart from being non-parametric, our proposed method has many other desirable properties. Our method is generic and even though we find very simple and symbolic methods to work very well for many KG datasets, every component of our model can be augmented by plugging in sophisticated neural models. For example, currently we use simple symbolic string matching to find if a relation path exists for the query entity. However, this component can be replaced with a neural model that matches paths which are semantically

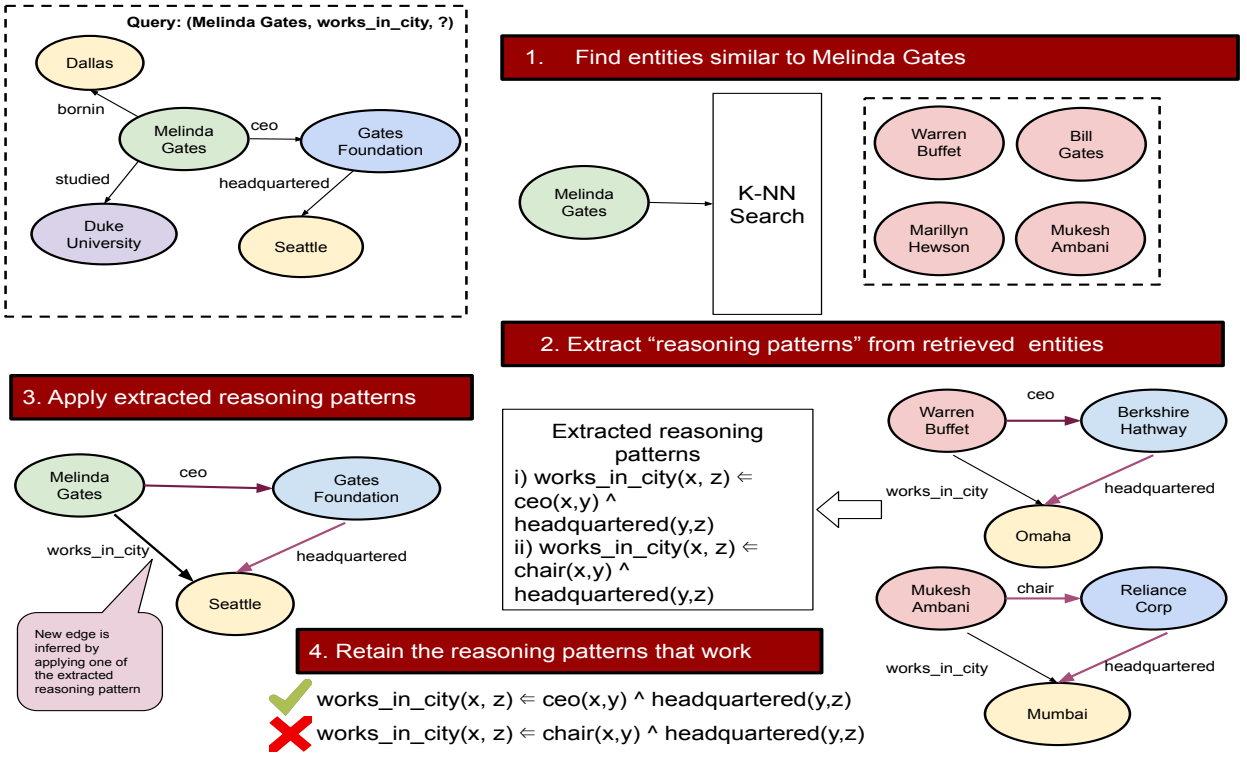


Figure 3.1: Overview of our approach. Given a query (Melinda, works_in_city, ?), our method first retrieves similar entities to the query entity. Then it gathers the reasoning paths that lead to the answer for the respective retrieved entities. The reasoning paths are applied for the query entity (Melinda) to retrieve the answer.

similar to each other Das et al. (2017). Similarly, we currently use a very simple inner product to compute similarities between entity embeddings. But that can be replaced with more sophisticated maximum inner product search Mussmann and Ermon (2016).

The contributions of the paper are as follows — (a) We present a non-parametric approach for reasoning over KGs, that uses *multiple* paths of evidence to derive an answer. These paths are gathered from entities in different parts of the KG that are similar to the query entity. (b) Our approach requires *no training* and can be readily applied to any new KGs. (c) Our method achieves state-of-the-art performance on NELL-995 and the harder subset of FB-122 outperforming sophisticated neural approaches and other tensor factorization methods. We also perform competitively on the WN18RR dataset. (d) Lastly, we provide detailed analysis about why our method outperforms parametric rule learning approaches like MINERVA Das et al. (2018).

3.2 Method

3.2.1 Notation and Task Description

Let $\mathcal{E}$ denote the set of entities and $\mathcal{R}$ denote the set of binary relations. A knowledge base (KB) is a collection of facts stored as triplets $(\mathrm{e}_1, \mathrm{r}, \mathrm{e}_2)$ where $\mathrm{e}_1, \mathrm{e}_2 \in \mathcal{E}$ and $\mathrm{r} \in \mathcal{R}$. From the KB, a knowledge graph $\mathcal{G}$ can be constructed where the entities $\mathrm{e}_1, \mathrm{e}_2$ are represented as the nodes and relation r as labeled edge between them. Formally, a KG is a directed labeled multigraph $\mathcal{G} = (V, E, \mathcal{R})$, where V and E denote the vertices and edges of the graph respectively. Note that $V = \mathcal{E}$ and $E \subseteq V \times \mathcal{R} \times V$. Also, following previous approaches Bordes et al. (2013); Xiong et al. (2017), we add the inverse relation of every edge, i.e. for an edge $(\mathrm{e}_1, \mathrm{r}, \mathrm{e}_2) \in E$, we add the edge $(\mathrm{e}_2, \mathrm{r}^{-1}, \mathrm{e}_1)$ to the graph. (If the set of binary relations $\mathcal{R}$ does not contain the inverse relation r^{-1}, it is added to $\mathcal{R}$ as well). A path in a KG between two entities e_s, e_t is defined as

a sequence of alternating entity and relations that connect $\mathrm{e_s}$ and $\mathrm{e_t}$. A length of a path is the number of relation (edges) in the path. Formally, let a path $\mathrm{p} = (\mathrm{e}_1, \mathrm{r}_1, \mathrm{e}_2, \ldots, \mathrm{r_n}, \mathrm{e}_{n+1})$ with $\mathrm{st(p)} = \mathrm{e}_1$, $\mathrm{en(p)} = \mathrm{e}_{n+1}$ and $\mathrm{len(p)} = \mathrm{n}$. Let $\mathcal{P}$ denote the set of all paths and $\mathrm{P_n}$ denote the set of all paths with length up to n, i.e. $\mathrm{P_n} \subseteq \mathcal{P} = \{p \in \mathcal{P} \mid \mathrm{len(p)} \leq \mathrm{n}\}$.

We consider the task of query answering on KGs. Query answering seeks to answer questions of the form $(\mathrm{e_{1q}}, r_q, ?)$ (e.g. Melinda, works_in_city, ?), where the answer is an entity in the KG.

3.2.2 Case-based Reasoning on Knowledge Graphs

This section describes how we apply case-based reasoning (CBR) in a KG. In CBR, given a new problem, similar cases are retrieved from memory Aamodt and Plaza (1994). In our setting, a problem is a query $(\mathrm{e_{1q}}, r_q, ?)$ for a missing edge in the KG. A case is defined as an observed fact $(\mathrm{e}_1, \mathrm{r}, \mathrm{e}_2)$ in a KG along with a set of paths up to length n that connect e_1 and e_2. Formally, a case $\mathrm{c} = (\mathrm{e}_1, \mathrm{r}, \mathrm{e}_2, \mathrm{P}) \subseteq V \times \mathcal{R} \times V \times \mathrm{P_n}$ is a 4 tuple where $(\mathrm{e}_1, \mathrm{r}, \mathrm{e}_2) \in \mathcal{G}$ and $\mathrm{P} \subseteq \mathrm{P}_{(\mathrm{e}_1, \mathrm{e}_2)} = \{\mathrm{p} \in \mathrm{P_n} \mid \mathrm{st(p)} = \mathrm{e}_1, \mathrm{en(p)} = \mathrm{e}_2\}$. In practice, it is intractable to store *all* paths between e_1 and e_2 ($\mathrm{P}_{(\mathrm{e}_1, \mathrm{e}_2)}$) and therefore we only store a random sample. Note, that we have a case for every observed triple in the KG and we denote the set of all such cases as $\mathcal{C}$. We also assume access to a pre-computed similarity matrix $\mathrm{S} \in \mathbb{R}^{V \times V}$ which stores the similarity between any two given entities in a KG. Intuitively, two entities which exhibit the same relations should have a high similarity (e.g., an athlete should have a high similarity score with another athlete). Lastly, we define a memory $\mathcal{M}$ that serves as the store for all the cases present in a KG and also the similarity matrix, i.e. $\mathcal{M} = (\mathcal{C}, \mathrm{S})$.

As explained earlier, CBR broadly comprises of four steps, which we explain briefly for our setup on a KG.

- **Retrieve**: In this step, given a query $(\mathrm{e_{1q}}, r_q, ?)$, our method first retrieves a set of k similar entities w.r.t $\mathrm{e_{1q}}$ using the pre-computed similarity matrix S, such that for each entity e' in the retrieved set, we observe at least one fact of the form $(\mathrm{e}', r_q, \mathrm{e}'')$ in $\mathcal{G}$. In other words, each retrieved entity should have at least one outgoing edge in $\mathcal{G}$ with the edge label as $\mathrm{r_q}$. Next, we gather all such facts $(\mathrm{e}', r_q, \mathrm{e}'')$ in $\mathcal{G}$ for each of the retrieved e'. Note, that there could be more than one fact for a given entity and query relation (e.g. USA, has_city, New York City and USA, has_city, Boston, for the entity 'USA' and query relation 'has_city'). Finally for all the gathered facts from the k-nearest neighbors, we retrieve all the cases from the memory store $\mathcal{M}$. As noted before, in our formulation, a case is a fact augmented with a sample of KG paths that connect the entities of the fact. As we will see later (§ 3.3), the KG paths often represent reasoning rules that determine why the fact $(\mathrm{e}', r_q, \mathrm{e}'')$ hold true. The goal of CBR is to reuse these rules for the new query.
- **Reuse**: The reasoning paths that were gathered in the retrieve step are re-used for the query entity. As described before, a path in a KG is an alternating sequence of entities and relations. The path gathered from the nearest neighbors have entities which are in the immediate neighborhood of the nearest neighbor entities. To reuse these paths, we first replace the entities from the paths with un-instantiated variables and only extract the sequence of relations in them Schoenmackers et al. (2010). For example, if we have a retrieved case (a fact with a set of paths) such as $(\mathrm{USA}, \mathrm{has\_city}, \mathrm{Boston})$, $\{(\mathrm{USA}, \mathrm{has\_state}, \mathrm{Massachusetts}, \mathrm{city\_i}$ we remove the entities from the path and extract rules such as: $\mathrm{has\_state(x, y)}, \mathrm{city\_in\_state(y, z)} \implies \mathrm{has\_city(x, z)}$. We gather all such paths from all the cases retrieved for a query. Since the same path type can occur in different cases, we maintain a list of paths sorted w.r.t the counts (in descending order).
- **Revise**: After the paths have been gathered, we look if those paths exists for the query entity $\mathrm{e_{1q}}$. In our approach we find that simple symbolic exact-string matching for the relations works quite well. However, neural relation extraction systems Zeng et al. (2014); Verga et al. (2016b) can be incorporated to map a relation to other similar relations to improve recall. We keep this direction as a part of our future work. Instead if we find an exact match for the sequence of relations, we revise the rules by instantiating the variables with the entities which lie along the path in the neighborhood of $\mathrm{e_{1q}}$.
- **Retain**: Finally, a case including the query fact and the paths that lead to the correct answer for the query can be added to the memory store $\mathcal{M}$.

Computing the similarity matrix: CBR approaches need access to a similarity matrix S to retrieve similar cases to the query entity. Intuitively, the similarity between entities that have similar relations should be higher (e.g. similarity between two athletes should be higher than the similarity between an athlete and a country). To model this, we parameterize each entity with an m-hot vector $\mathbf{e} \in \mathbb{R}^{\mathcal{R}}$. That is, each entity is a m-hot vector with the dimension equal to the size of the number of binary relations in the KB. An entry in the vector is set to 1, if an entity has at least one edge with that relation type, otherwise is set to 0. Even though, this is a really simple way of parameterizing an entity, we found this to work extremely well in practice. However, as previously mentioned we propose a generic method and one could replace the entity embeddings with any pre-trained vectors from any model. As an example, we present experiments by replacing our m-hot representation with pre-trained embeddings obtained from the state-of-the-art RotatE model Sun et al. (2019b). Lastly, the similarity between two entities is calculated by a simple inner product between the normalized embeddings.

Caching 'cases' in the memory store: CBR also needs access to store containing cases. As mentioned before, in our setup a 'case' is a KG triple, along with a sample of paths that connect the two entities of the triple. Since the number of paths between entities grow exponentially w.r.t path length, it is intractable to store all paths between an entity pair. We instead consider a small subgraph around each entity in the KG spanned by 1000 randomly sampled paths of length up to 3. Next, for each triple $(\mathrm{e}_1, \mathrm{r}, \mathrm{e}_2)$ in the KG, we exhaustively search the subgraph around e_1, collected in the previous step to find paths up to length 3 which lead to e_2. These paths along with the fact form a case. This process is repeated for all the triples and each case is added to the case store $\mathcal{C}$.

Both the similarity matrix S and the case store $\mathcal{C}$ are pre-computed offline and is stored in the memory $\mathcal{M}$. Once that is done, our method requires *no further training* and can be readily used for any query in the KG.

3.3 Experiments

3.3.1 Data and Evaluation Protocols

We test our CBR based approach on three datasets that are often used in the community for benchmarking models — FB15K-237 Guo et al. (2016), NELL-995 Xiong et al. (2017), and WN18RR Dettmers et al. (2018b). FB15K-237 comes with a set of KB rules that can be used to infer missing triples in the dataset. We do not use the rules in the dataset and even show that our model is able to recover the rules from similar entities to the query. WN18RR was created by Dettmers et al. (2018b) from the original WN18 dataset by removing various sources of test leakage, making the datasets more realistic and challenging. We compare our CBR based approach with various state-of-the-art models using standard ranking metrics such as HITS@N and mean reciprocal rank (MRR). For fair comparison to baselines, after a fact is predicted we do not add the new case (inferred fact and paths) in the memory (retain step in § 3.2.2), as that would mean we would use more information than our baselines to predict the followup queries.

Hyper-parameters: The various hyper-parameters for our method are the number of nearest neighbor retrieved for a query entity (k), the number of paths that are gathered from the retrieved entity (l) and the maximum path length considered (n). For all our experiments, we set $n = 3$. We tune k and l for each dataset w.r.t the given validation set.

3.3.2 Results on Query Answering

We first present results on query answering (link prediction) on the three datasets and compare to various state-of-the-art baseline models. It is quite common in literature Bordes et al. (2013); Yang et al. (2015); Dettmers et al. (2018b); Sun et al. (2019b) to report aggregate results on both tail prediction $(\mathrm{e}_1, \mathrm{r}, ?)$ and head prediction $(?, \mathrm{r}^{-1}, \mathrm{e}_2)$. To be exactly comparable to baselines, we report results on tail prediction for NELL-995 and for other datasets, we report average of head and tail predictions.

NELL-995: Table 3.2 reports the query answering performance on the NELL-995 dataset. We compare to

	Model	HITS@3	HITS@5	HITS@10	MRR
WITH RULES	KALE-Pre Guo et al. (2016)	0.358	0.419	0.498	0.291
	KALE-Joint Guo et al. (2016)	0.384	**0.447**	**0.522**	0.325
	ASR-DistMult Minervini et al. (2017)	0.363	0.403	0.449	0.330
	ASR-ComplEx Minervini et al. (2017)	0.373	0.410	0.459	0.338
WITHOUT RULES	TransE Bordes et al. (2013)	0.360	0.415	0.481	0.296
	DistMult Yang et al. (2017)	0.360	0.403	0.453	0.313
	ComplEx Trouillon et al. (2016)	0.370	0.413	0.462	0.329
	GNTPs Minervini et al. (2020)	0.337	0.369	0.412	0.313
	CBR (Ours)	**0.424**	**0.471**	**0.515**	**0.378**

Table 3.1: Link prediction results on the FB15K-237 dataset.

Metric	ComplEx	ConvE	DistMult	MINERVA	CBR
HITS@1	0.612	0.672	0.610	0.663	**0.705**
HITS@3	0.761	0.808	0.733	0.773	**0.828**
HITS@10	0.827	0.864	0.795	0.831	**0.875**
MRR	0.694	0.747	0.680	0.725	**0.772**

Table 3.2: Query-answering results on NELL-995 dataset.

several strong baselines. In particular, we compare to various embedding based models such as DistMult Yang et al. (2015), ComplEx Trouillon et al. (2016) and ConvE Dettmers et al. (2018b). We also wanted to compare to various neural models for learning logical rules such as neural-theorem-provers Rocktäschel and Riedel (2017b), NeuralLP Yang et al. (2017) and MINERVA Das et al. (2018). However, only MINERVA scaled to the size of NELL-995 dataset and others did not. As it is clear from table 3.2, CBR outperforms all the baselines by a large margin with gains of over 4% on the strict HITS@1 metric. We further discuss and analyze the results in sec (3.3.4).

FB15K-237: Next we consider the FB15K-237 dataset by Guo et al. (2016). Comparing results on FB15K-237 is attractive for a couple of reasons — (a) Firstly, this dataset comes with a set of logical rules hand coded by the authors that can be used for logical inference. It would be interesting to see if our CBR approach is able to automatically uncover the rules from the data. (b) Secondly, there is a recent work on a neural model for logical inference (GNTPs) Minervini et al. (2020) that scales neural-theorem-provers Rocktäschel and Riedel (2017b) to this dataset and hence we can directly compare with them. Table 3.1 reports the results. We compare with several baselines. CBR significantly outperforms GNTPs and even outperforms most models which have access to the hand-coded rules during training. We also find that CBR is able to uncover correct rules for 27 out of 31 (87%) query relations.

WN18RR: Finally, we report results of WN18RR in table 3.3. CBR performs competitively with GNTPs and most embedding based methods except RotatE Sun et al. (2019b). Upon further analysis, we find that for 210 triples in the test set, the entity was not present in the graph and hence no answers were returned for those query entities.

Metric	TransE	DistMult	ComplEx	ConvE	RotatE	GNTP	CBR
HITS@1	-	0.39	0.41	0.40	**0.43**	0.41	0.39
HITS@3	-	0.44	0.46	0.44	**0.49**	0.44	0.46
HITS@10	0.50	0.49	0.51	0.52	**0.57**	0.48	0.51
MRR	0.23	0.43	0.44	0.43	**0.48**	0.43	0.43

Table 3.3: Link prediction results on WN18RR dataset.

Model	HITS@1	HITS@10	MRR
NeuralLP Yang et al. (2017)	0.048	0.351	0.179
NTP-λ Rocktäschel and Riedel (2017b)	0.102	0.334	0.155
MINERVA Das et al. (2018)	0.162	0.283	0.201
MultiHop(DistMult) Lin et al. (2018a)	0.145	0.306	0.200
MultiHop(ConvE) Lin et al. (2018a)	0.178	0.329	0.231
Meta-KGR(DistMult) Lv et al. (2019)	0.197	0.345	0.248
Meta-KGR(ConvE) Lv et al. (2019)	0.197	0.347	0.253
CBR (ours)	**0.234**	**0.403**	**0.293**

Table 3.4: Link prediction results on NELL-995 for few shot relations.

3.3.3 Experiments with limited data

As mentioned before, our CBR based approach needs no training and gathers reasoning patterns from few similar entities. Therefore, it should ideally perform well for query relations for which we do not have a lot of data. Recently, Lv et al. (2019) studied this problem and propose a meta-learning Finn et al. (2017) based solution (Meta-KGR) for few-shot relations. We compare with their model to see if CBR based approach will be able to generalize for such few-shot relations. Table 3.4 reports the results and quite encouragingly we find that we outperform all sophisticated meta-learning approaches by a large margin.

3.3.4 Analysis: CBR is capable of doing contextualized reasoning

In this section, we analyze the performance of our non-parametric approach and try to understand why it works better than existing parametric rule learning models like MINERVA. Lets consider the relation 'agent_belongs_to_organization' in the NELL-995 dataset. Here the query entity can belong to a wide varity of types. For example, these are all triples for the query relation in NELL-995— (George Bush, agent_belongs_to_organization, House of Republicans), (Vancouver Canucks, agent_belongs_to_organization, NHL), (Chevrolet, agent_belongs_to_organization, General Motors). As can be seen, the query entity for a relation can belong to many different types and hence the logical rules that needs to be learned would be different. An advantage of CBR based approach is that, for each query entity it retrieves similar contexts and then gather rules from them. One the other hand, models like MINERVA has to encode all rules into its parameters which makes learning harder. In other words, CBR is capable of doing better fine-grained contextual reasoning for a given query. To further confirm this hypothesis, we count the number of paths that CBR finds that lead to the correct answer and compare it with MINERVA. CBR learns a total of 306.4 unique paths that lead to answer compared to 176.83 of MINERVA. Figure 3.2 plots the counts for each query relation which further shows that CBR finds more varied paths than MINERVA.

3.3.5 Results with RotatE embeddings

As mentioned before, the CBR approach is generic and one can incorporate sophisticated models into each of the step. We run experiments where the m-hot representation of entities is replaced with pretrained embeddings obtained from a trained RotatE Sun et al. (2019b) model for building the similarity matrix. On the WN18RR dataset, we get a MRR of 0.425 as compared to 0.423 with our original approach.

3.3.6 Limitations of our Current Work

A key limitation of our current work is the symbolic matching of reasoning paths. Different symbolic sequence of relation can have similar semantics (e.g. works_in_org, org_located_in and studies_in, college_located_in are similar paths for inferring lives_in relation), but in our current work, these paths would be treated differently. This limitation can be alleviated by learning a similarity kernel for paths using distributed representation of relations.

On error analysis, we found that a major source of error occurs in our ranking of entities. Currently, ranking of predicted entities is done via number of paths that lead to it. Even though, this simple technique works

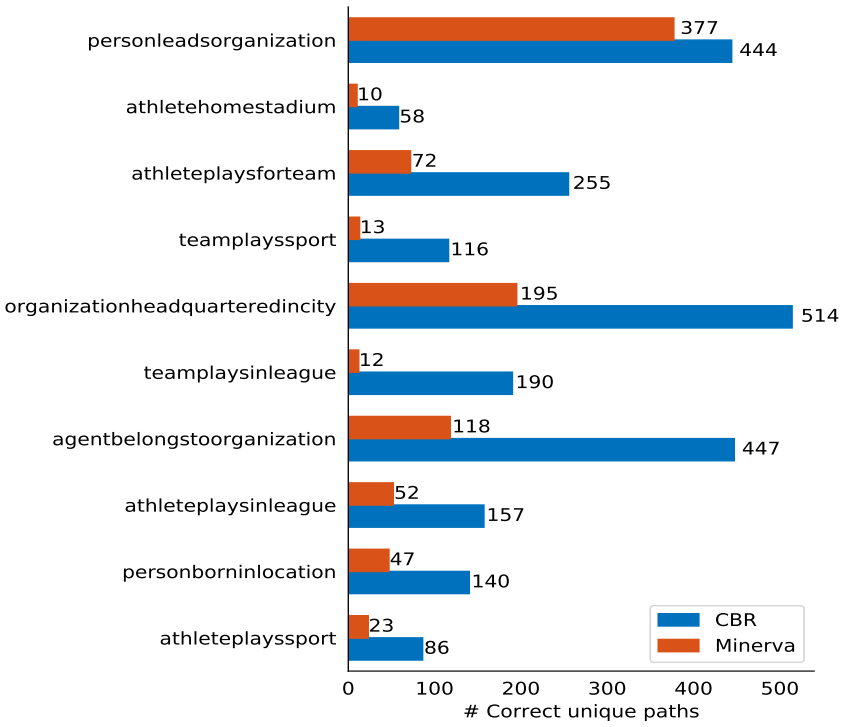


Figure 3.2: The number of unique correct paths CBR and MINERVA find for each query relation in NELL-995

well, we noticed that, if we had access to an oracle ranker, our performance would improve significantly. For example, in FB15K-237 dataset, CBR retrieves a total of 241 entities out of more than 9.5K entities. An oracle ranking of these entities would increase the accuracy (HITS@1) from 0.28 to 0.74. This indicates that a substantial gain could be obtained if we train a model to rank the retrieved entities, a direction we leave as future work.

3.3.7 Inference time

Table 3.5 report the inference times of our model on the entire evaluation set of WN18RR (6268 queries) and NELL-995 (2825 queries) and compares it with MINERVA. Since our approach first retrieves similar entities and then gathers reasoning paths from them, it is slower than MINERVA. However, given the empirical improvements in accuracy, we believe this is not a significant tradeoff.

Dataset	MINERVA	CBR
WN18RR	63s	69s
NELL-995	35s	68s

Table 3.5: Inference time (in seconds) on two datasets.

3.4 Related Work

Bayesian non-parametric approaches for link prediction: There is a rich body of work which employs bayesian non-parametric approaches to automatically learn the latent dimension of entities. The infinite relational model (IRM) Kemp et al. (2006) and its extensions Xu et al. (2006) learns a possibly unbounded number of latent clusters from the graph and an entity is represented by its cluster membership. Later Airoldi et al. (2008) proposed the mixed membership stochastic block models that allows entities to have mixed membership over the clusters. Instead of cluster membership, Miller et al. (2009) propose a model that learn features of entities and the non-parametric approach allows learning unbounded number of dimensions. Sutskever et al. (2009) combine bayesian non-parametric and tensor factorization approaches and Zhu et al. (2012) allows non-parametric learning in a max-margin framework. Our method does not

learn latent dimension and features of entities using a bayesian approach. Instead we propose a framework for doing non-parametric reasoning by learning patterns from k-nearest neighbors of the query entity. Also, the bayesian models have only been applied to very small knowledge graphs (containing few hundred entities and few relations).

Rule induction in knowledge graphs is a very rich field with lots of seminal works. Inductive Logic Programming (ILP) Muggleton et al. (1992) learns general purpose predicate rules from examples and background knowledge. Early work in ILP such as FOIL Quinlan (1990), PROGOL Muggleton (1995) are either rule-based or require negative examples which is often hard to find in KBs (by design, KBs store true facts). Statistical relational learning methods Getoor and Taskar (2007); Kok and Domingos (2007); Schoenmackers et al. (2010) along with probabilistic logic Richardson and Domingos (2006); Broecheler et al. (2010); Wang et al. (2013) combine machine learning and logic but these approaches operate on symbols rather than vectors and hence do not enjoy the generalization properties of embedding based approaches. Moreover, unlike our approach, these methods do not learn rules from entities similar to the query entity. Recent work in rule induction, as discussed before Yang et al. (2017); Rocktäschel and Riedel (2017b); Das et al. (2018); Minervini et al. (2020) try to encode rules in the parameters of the model. In contrast, we propose a non-parametric approach for doing so. Moreover our model outperforms them on several datasets.

K-NN based approach in other NLP applications: Nearest neighbor models have been applied to a number of NLP applications in the past such as parts-of-speech tagging Daelemans et al. (1996) and morphological analysis Bosch et al. (2007). There has also been several recent work which leverages k-nearest neighbors for various NLP tasks, which is a step towards case based reasoning. Retrieve-and-edit based approaches are gaining popularity for various structured prediction tasks Guu et al. (2018); Hashimoto et al. (2018). Accurate sequence labeling by explicitly and only copying labels from retrieved neighbors have been achieved by Wiseman and Stratos (2019). Another recent line of work use training examples at test time to improve language generation Weston et al. (2018); Pandey et al. (2018); Cao et al. (2018); Peng et al. (2019). Improvements in language model have been also observed by Khandelwal et al. (2020) by utilizing explicit examples from past training data obtained from nearest neighbor search in the encoded space. However, unlike us these work do not extract explicit reasoning patterns (or solutions in cases) from nearest neighbors.

3.5 Conclusion

We propose a very simple non-parametric approach for reasoning in KGs that is similar to case-based reasoning approaches in classical AI. Our proposed model requires no training and can be readily applied to any knowledge graphs. It achieves new state-of-the-art performance in NELL-995 and FB15K-237 datasets. Also we show that, our approach is robust in low-data settings. Overall, our non-parametric approach is capable of deriving crisp logical rules for each query by extracting reasoning patterns from other entities and entirely removes the burden of storing logical rules in the model parameters.

3.6 Probabilistic Case-based Reasoning over Knowledge Bases

3.6.1 Introduction

We live in an evolving world with a lot of heterogeneity as well as new entities being created continuously. For example, scientific papers and Wikipedia pages describing facts about new entities, are being constantly added (e.g. COVID-19). These new findings further trigger the inference of newer facts, each with its own diverse reasoning. We are interested in developing such automated reasoning systems for large knowledge-bases (KBs).

In machine learning, non-parametric methods hold the promise of handling evolving data Cover and Hart (1967); Rasmussen (2000). Most current KG completion models learn low dimensional parametric representation of entities and relations via tensor factorization or sophisticated neural approaches Nickel et al. (2011b); Bordes et al. (2013); Socher et al. (2013); Sun et al. (2019b); Vashishth et al. (2020). Another line of work learns Horn-clause style reasoning rules from the KG and stores them in its parameters Rocktäschel and Riedel (2017b); Das et al. (2018); Minervini et al. (2020). However, these parametric approaches work with a fixed set of entities and it is unclear how these models will adapt to new entities.

This paper presents a k-nearest neighbor (KNN) based approach for KG reasoning that is reminiscent of case-based reasoning (CBR) in classical AI. A CBR system solves a new problem by retrieving 'cases' that are similar to the given problem, revising the solution to retrieved cases (if necessary) and reusing it for the new problem (Schank, 1982; Leake, 1996, inter-alia). For the task of finding a target entity given a source entity and binary KG relation (e.g. (JOHN VON NEUMAN, PLACE_OF_DEATH, ?) in Figure 3.3), our approach first retrieves k similar entities (cases) to the query entity. Next, for each retrieved entity, it finds multiple KG paths[1] (each path is a solution to retrieved cases) to the entity they are connected by the query relation (e.g. paths between (RICHARD FEYNMAN, USA)). However, one solution seldom works for all queries. For example, even though the path 'BORN_IN' is predictive of 'PLACE_OF_DEATH' for US-born scientists (figure 3.3), it does not work for scientists who have immigrated to USA. To handle this, we present a probabilistic CBR approach which learns to weighs paths with respect to an estimate of its prior and its precision, given the query. The prior of a path represents its frequency while the precision represents the likelihood that the path will lead to a correct answer entity. To obtain robust estimates of the path parameters, we cluster similar entities together and compute them by simple count statistics (§3.6.2).

Apart from computing these estimates, our method needs *no further training*. Overall, our simple approach outperforms several recent parametric rule learning methods Das et al. (2018); Minervini et al. (2020) and performs competitively with various state-of-the-art KG completion approaches Dettmers et al. (2018b) on multiple datasets.

An advantage of non-parametric models is that it can adapt to growing data by adjusting its number of parameters. In the same spirit, we show that our model can seamlessly handle an 'open-world' setting in which *new entities* arrive in the KG. This is made possible by several design choices such as (a) representing entities as sparse (non-learned) vector of its relation types (§3.6.1), (b) our use of an online non-parametric hierarchical clustering algorithm Monath et al. (2019) that can efficiently recompute changes in cluster assignments because of the newly added entity (§3.6.2), (c) and a simple and efficient way of recomputing the prior and precision parameters for paths per cluster (§3.6.2).

Current models for KG completion that learn entity representations for a fixed set of entities cannot handle the open-world setting. In fact we show that, retraining the models continually with new data leads to severe degradation of the model performance with models forgetting what it had learned before. For example, the performance (MRR) of ROTATE model Sun et al. (2019b) drops by 11 points (absolute) on WN18RR in this setting (§3.6.3). On the other hand, we show that with new data, the performance of our model is consistent as it is able to seamlessly reason with the newly arrived data.

Our work is most closely related to a recent concurrent work by **?** where they propose a model that gathers paths from entities similar to the query entity. However, **?** encourages path that occur frequently in the KG and does not learn to weigh paths differently for queries. This often leads to wrong inference leading to

[1] A path is a contiguous sequence of KG facts such as RICHARD FEYNMAN → AFFILIATED → CALTECH → LOCATED → USA.

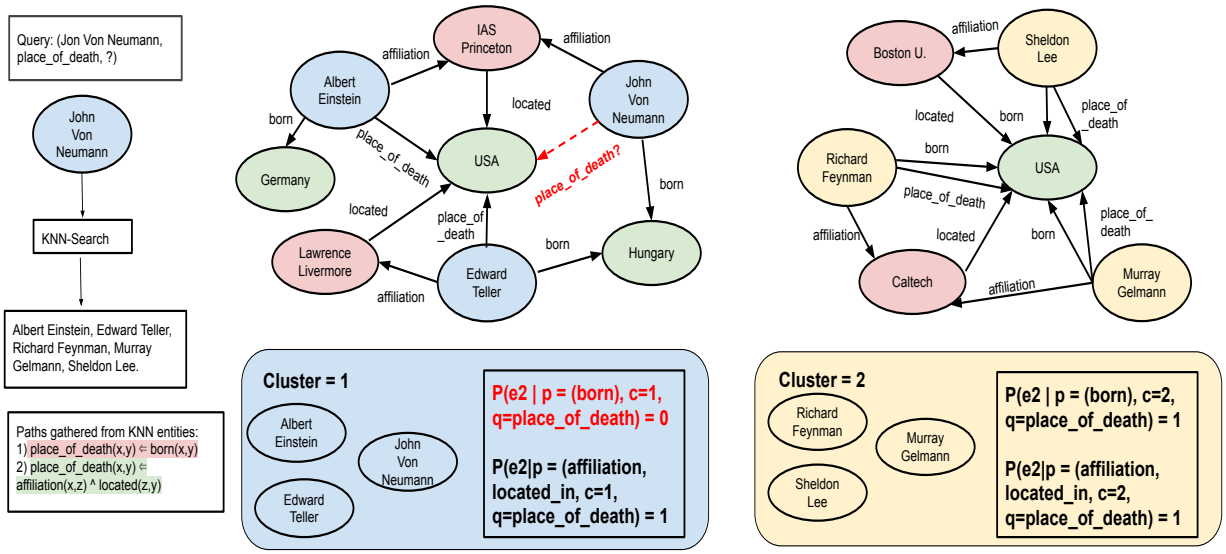


Figure 3.3: Given the query, (JON VON NEUMANN, PLACE_OF_DEATH, ?), our model gathers reasoning paths from similar entities such as other scientists. However, not all gathered paths work for a query e.g. the path ('BORN(x, y)') would not work for VON NEUMANN. This highlights the importance of learning path weights for *clusters of similar entities*. Even though 'BORN_IN' could be a reasonable path for predicting PLACE_OF_DEATH, this does not apply for VON NEUMANN and other scientists in his cluster. The precision parameter of the path given the cluster helps in penalizing the 'BORN_IN' path. Note that the node USA is repeated twice in the figure to reduce clutter.

low performance. For example, on the test-II evaluation subset of FB122 where all triples can be inferred by logical rules, **?** scores quite low (63 MRR) because of learning incorrect rules. On the other hand, we score significantly higher (94.83 MRR) demonstrating that we can learn more effective rules. In fact, we consistently and significantly outperform **?** on several benchmark datasets. Also, unlike us, they do not test themselves in the challenging open-world setting.

The contributions of this paper are as follows: (a) We present a KNN based approach for KG completion that gathers reasoning paths from entities that are similar to the query entity. Following a principled probabilistic approach (§3.6.1), our model weighs each path by its likelihood of reaching a correct answer which penalizes paths that are spurious in nature. (b) The parameters of our model grow with data and can be estimated efficiently using simple count statistics (§3.6.2). Apart from this, our approach needs *no training*. We show that our simple approach significantly outperforms various rule learning methods Das et al. (2018); Minervini et al. (2020); **?** on many benchmark datasets. (c) We also show that our model can easily handle addition of facts about new entities and is able to seamlessly integrate and reason with the newly added data significantly outperforming parametric embedding based models.

Notation and Task Description

Let $\mathcal{V}$ denote the set of entities, $\mathcal{R}$ denote the set of binary relations and $\mathcal{G}$ denote a KB or equivalently a Knowledge Graph (KG). Formally, $\mathcal{G} = (\mathcal{V}, E, \mathcal{R})$ is a directed labeled multigraph where $\mathcal{V}$ and E denote the vertices and edges of the graph respectively. Note that, $E \subseteq \mathcal{V} \times \mathcal{R} \times \mathcal{V}$. Let (e_1, r, e_2) denote a fact in $\mathcal{G}$ where $e_1, e_2 \in V$ and $r \in E$. Also, following previous approaches Bordes et al. (2013), we add the inverse relation of every edge, i.e., for an fact $(e_1, r, e_2) \in E$, we add the edge (e_2, r^{-1}, e_1) to the graph. (If the set of binary relations $\mathcal{R}$ does not contain the inverse relation r^{-1}, it is added to $\mathcal{R}$ as well).

Task: We consider the task of query answering on KGs, i.e., answering questions of the form $(e_{1q}, r_q, ?)$, where answer is an entity in the KG.

Paths in KG: A path in a KG between two entities e_s, e_t is defined as a sequence of alternating entity and relations that connect e_s and e_t. A length of a path is the number of relation (edges) in the path. Formally, let a path $p = (e_1, r_1, e_2, \ldots, r_n, e_{n+1})$ with $\mathsf{st}(p) = e_1$, $\mathsf{en}(p) = e_{n+1}$ and $\mathsf{len}(p) = n$. We also define a *path type* as the sequence of the relations in p, i.e., $\mathsf{type}(p) = (r_1, r_2, \ldots, r_n)$. Let $\mathcal{P}$ denote the set of all paths in $\mathcal{G}$. Let $\mathcal{P}_n \subseteq \mathcal{P} = \{p \mid \mathsf{len}(p) \leq n\}$ be the set of all paths of length up to n. Also, let $\mathrm{P_n}$ denote the set of all path *types* with length up to n, i.e. $\mathrm{P_n} = \{\mathsf{type}(p) \mid p \in \mathcal{P}_n\}$. Let $\mathrm{P_n}(e_1, r) \subseteq \mathrm{P_n}$ denote all path types of length up to n that originate at e_1 and end at the entities that are connected to e_1 by a direct edge of type r. In other words, if $S_{e_1 r} = \{e_2 \mid (e_1, r, e_2) \in \mathcal{G}\}$ denotes the set of entities that are connected to e_1 via a direct edge r, then $\mathrm{P_n}(e_1, r)$ denotes the set of all path types of length up to n that start from e_1 and end at entities in $S_{e_1 r}$. By definition, $r \in \mathrm{P_n}(e_1, r)$. Similarly, we define $\mathcal{P}_n(e_1, r)$ which contain paths instead of path types.

Model

Given a query, our approach gathers KG path types from entities that are similar to the query entity. Each path type is weighed with respect to an estimate of both its frequency and precision (§3.6.1). By clustering similar entities together (§3.6.2), our model obtains robust estimate of the path statistics (§3.6.2). Our approach is non-parametric because - (a) Instead of storing reasoning rules in parameters Das et al. (2018); Minervini et al. (2020), it derives them dynamically from k-similar entities (like a non-parametric k-nn classifier Cover and Hart (1967)). (b) We cluster entities together using a non-parametric clustering approach and provide an efficient way of adding / estimating parameters when entities are added to the KG (§3.6.2).

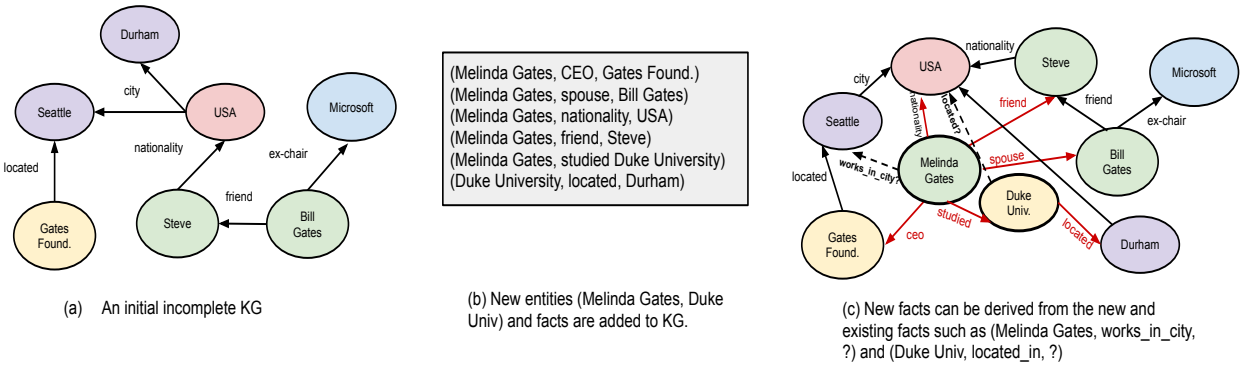
Reasoning from contextual entities

Our approach first finds k similar entities to the query entity that have atleast an edge of type r_q. For example, for the query (MELINDA GATES, WORKS_IN_CITY, ?), we would consider WARREN BUFFET if we observe (WARREN BUFFET, WORKS_IN_CITY, OMAHA). We refer to these entities as 'contextual entities'. Each entity is represented as a sparse vector of its outgoing edge types, i.e. $\mathbf{e_i} \in \{0, 1\}^{|\mathcal{R}|}$. If entity e_i has m distinct outgoing edge types, then the dimension corresponding to those types are set to 1. This is an extremely simple and flexible way of representing entities which we find to work well. Also note that, as more data is added about an entity, this sparse representation makes it trivial to update the embeddings.

Let $E_{c,q}$ denote the set of contextual entities for the query q. To compute $E_{c,q}$, we first sort entities with respect to their cosine distance with respect to query entity and select the k entities with the least distance and which have the query relation r_q. For each contextual entity e_c, we gather the path types (up to length n) that connect e_c to the entities it is connected by the edge r_q (i.e. $\mathrm{P_n}(e_c, r_q)$ in §3.6.1). These extracted path types will be used to reason about the query entity. Let $\mathrm{P_n}(E_{c,q}, r_q) = \bigcup_{e_c \in E_{c,q}} \mathrm{P_n}(e_c, r_q)$ represent the set of unique path types from the contextual entities. The probability of finding the answer entity e_2 given the query is given by:

$$
\begin{aligned}
P\left(e_2 \mid e_{1q}, r_q\right) &= \sum_{p \in \mathrm{P_n}(E_{(c,q)}, r_q)} P(e_2, p \mid e_{1q}, r_q) \\
&= \sum_{p} P(p \mid e_{1q}, r_q) P(e_2 \mid p, e_{1q}, r_q) \qquad (3.1)
\end{aligned}
$$

We marginalize the random variable representing the path types obtained from $E_{c,q}$. $P(p \mid e_{1q}, r_q)$ denotes the probability of finding a path type given the query. This term captures how frequently each path type co-occurs with a query and represents the prior probability for a path type. On the other hand, $P(e_2 \mid p, e_{1q}, r_q)$ captures the proportion of times, when a path type p is traversed starting from the query entity, we reach the correct answer instead of some other entity. This term can be understood as capturing the likelihood of reaching the right answer or the 'precision' of a reasoning path type. This is crucial in penalizing 'spurious' path types that sometimes coincidentally find the right answer entity. For example, for the query relation WORKS_IN_CITY, the path type (FRIEND∧ LIVES_IN_CITY) might have a high prior probability (since people often have many friends in the city where they work). However, this path is 'spurious' with respect to WORKS_IN_CITY, since they might have friends living in various cities and hence this path type will not necessarily return the correct answer.



(a) An initial incomplete KG

(b) New entities (Melinda Gates, Duke Univ) and facts are added to KG.

(c) New facts can be derived from the new and existing facts such as (Melinda Gates, works_in_city, ?) and (Duke Univ, located_in, ?)

Figure 3.4: We consider a setting where *new entities* and facts are added continuously to the KG. Our non-parametric approach can seamlessly reason with the newly added entities and can infer new facts about them (e.g. (MELINDA, WORKS_IN_CITY, ?) or (DUKE UNIV., LOCATED_IN_COUNTRY, ?)) without requiring expensive training.

3.6.2 Entity Clustering

Equation 3.1 has parameters for each entity in the KG. For large KGs, this can quickly lead to parameter explosion. Also, estimating per-entity parameter leads to noisy estimates due to sparsity. Instead, we choose to cluster similar entities together. Let c be a random variable representing the cluster assignment of the query entity. Then for the path-prior term, we have

$$P(p \mid e_{1q}, r_q) = \sum_c P(c \mid e_{1q}, r_q) P(p \mid c, e_{1q}, r_q)$$

We assume that each entity is assigned to one cluster, so $P(c \mid e_{1q}, r_q)$ is zero for all clusters except the cluster in which the query entity belongs to. Secondly we assume, that the prior probability of a path given the entity and cluster can be determined from the cluster alone and is independent of each entity in the cluster. In other words, if $c_{e_{1q}}$ is the cluster in which the $e_{1,q}$ has been assigned, then $P(p \mid c_{e_{1q}}, e_{1q}, r_q) = P(p \mid c_{e_{1q}}, r_q)$. Instead of per-entity parameters, we now aggregate statistics over entities in the same cluster and have per-cluster parameters. We also show that this leads to significantly better performance (§3.6.3). A similar argument applies for the path-precision term in which we calculate the proportion of times, a path leads to the correct answer entity starting from each entity in the cluster.

To perform clustering, we use hierarchical agglomerative clustering with average linkage with the entity-entity similarity defined in §3.6.1. We extract a non-parameteric number of clusters from the hierarchy using a threshold on the linkage function. Agglomerative clustering has been shown to be effective in many knowledge-base related tasks such as entity resolution Lee et al. (2012); Vashishth et al. (2018) and in general has shown to outperform flat clustering methods such as K-means Green et al. (2012); Kobren et al. (2017). A flat clustering is extracted from the hierarchical clustering by using a threshold on the linkage function score. We perform a breadth first search from the root of the tree stopping at nodes for which the linkage is above the given threshold. The nodes where the search stops give a flat clustering (refer to §**??** for more detail on this).

Parameter Estimation

Next we discuss how to estimate path prior and precision terms. There exists abundant modeling choices to estimate them. For example, following Chen et al. (2018), we could train a neural network model to estimate $P(p \mid c_{e_{1q}}, r_q)$. However, with our original goal of designing a simple and efficient non-parametric model, we estimate these parameters by simple count statistics from the KG. E.g., the path prior $P(p \mid c, r_q)$ is estimated

as

$$\frac{\sum_{e_c \in c} \sum_{p' \in \mathcal{P}_n(e_c, r_q)} \mathbb{1}\left[\mathsf{type}(p') = p\right]}{\sum_{e_c \in c} \sum_{p' \in \mathcal{P}_n(e_c, r_q)} \mathbb{1}} \tag{3.2}$$

For each entity in cluster c, we consider the paths that connect e_c to entities it is directly connected to via edge type r_q ($\mathcal{P}_n(e_c, r_q)$ in §3.6.1). The path prior for a path type p is computed as the proportion of times the type of paths in $\mathcal{P}_n(e_c, r_q)$ is equal to p. Note that in equation 3.2, if a path type appears multiple times, we count all instances. For example, for the query relation WORKS_IN_CITY, a path of the form (CO_WORKER $\wedge$ WORKS_IN_CITY) can occur multiple times, since a person can have multiple different co-workers. Considering just path types will lead to under-weighing of such important paths. Similarly, the path-precision probability ($P(e_2 \mid p, c, r_q)$) can be estimated as,

$$\frac{\sum_{e_c \in c} \sum_{p' \in \mathcal{P}_n(e_c)} \mathbb{1}[\mathsf{type}(p') = p] \cdot \mathbb{1}[\mathrm{en}(p') \in S_{e_c r_q}]}{\sum_{e_c \in c} \sum_{p' \in \mathcal{P}_n(e_c)} \mathbb{1}[\mathsf{type}(p') = p]} \tag{3.3}$$

Let $\mathcal{P}_n(e_c)$ denote the paths of up to length n starting from the entity e_c. Note, unlike $\mathcal{P}_n(e_c, r_q)$, the paths in $\mathcal{P}_n(e_c)$ do not have to end at specific entities. Also from §3.6.1, $\mathrm{en}(p)$ denotes the end entity for a path p and $S_{e_c r_q}$ denotes the set of entities that are connected to e_c via a direct edge of type r_q. Equation 3.3, therefore, estimates the proportion of times the path p successfully ends at one of the answer entities when starting from e_c, given r_q.

There are several advantages in estimating the parameters using simple count statistics. Firstly, they are extremely simple, and statistics for each entity in clusters can be computed in parallel making them extremely time efficient. Secondly once they are computed, our approach needs *no further training*. Lastly, when new data is added, it makes it easy to update the parameters without training from scratch.

To summarize, given a query entity (e_{1q}, r_q), our method gathers reasoning paths from k similar entities to e_{1q}. These reasoning paths are then traversed in the KG starting from e_{1q}, leading to a set of candidate answer entities. The score of each answer entity candidate is computed as a weighted sum of the reasoning paths the lead to them (Equation 3.1). Each path is weighed with an estimate of its frequency (Equation 3.2) and precision (Equation 3.3) given the query relation. The next section describes how we extend our model for open-world setting where new entities and facts are added to the KB.

Open-world Setting

A great benefit of non-parametric models is that it can seamlessly handle growing data by adding new parameters. New entities constantly arrive in the world (e.g. new Wikipedia articles about entities are frequently created). We consider a setting (Figure 3.4) in which new entities with few facts (edges) about them keep getting added to the KG. This setting is challenging for parametric models Das et al. (2018); Sun et al. (2019b) as it is unclear how these models can incorporate new entities without retraining from scratch. However, retraining to obtain entity embeddings on industrial scale KGs might be impractical (e.g. consider Facebook social graph where new users are joining continuously). Next, we show that our approach can handle this setting efficiently in the following way:
(a) **Adding/updating entity representations**: First we need to create entity representations for the newly arrived entities. Also, for some existing entities for which new edges were added (e.g. BILL GATES, DURHAM, etc. in figure 3.4), their representations need to be updated. Recall, that we represent entities as a sparse vector of its edge types and hence this step is trivial for our approach.
(b) **Updating cluster assignments**: Next the new entities needs to be added to clusters of similar entities. Also, the cluster assingments of entities that got updated can also change as well and their change can further trigger changes to the clustering of other entities. To handle this, one could naively cluster all entities in the KG, however that could be wasteful and time-consuming for large KGs. Instead, we use an online hierarchical clustering algorithm - GRINCH Monath et al. (2019), which has shown to perform as well as agglomerative clustering in the online setting. GRINCH observes one entity at a time, placing it next to its

	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{R}\rvert$	$\lvert E\rvert$
NELL-995	75,492	200	154,213
FB122	9,738	122	112,476
WN18RR	40,943	11	93,003

Table 3.6: Dataset Statistics

		Test-I				Test-II				Test-ALL			
		Hits@N (%)			MRR	Hits@N (%)			MRR	Hits@N (%)			MRR
		3	5	10		3	5	10		3	5	10	
With Rules	KALE-Pre Guo et al. (2016)	35.8	41.9	49.8	0.291	82.9	86.1	89.9	0.713	61.7	66.2	71.8	0.523
	KALE-Joint Guo et al. (2016)	38.4	44.7	52.2	0.325	79.7	84.1	89.6	0.684	61.2	66.4	72.8	0.523
	ASR-DistMult Minervini et al. (2017)	36.3	40.3	44.9	0.330	98.0	99.0	99.2	0.948	70.7	73.1	75.2	0.675
	ASR-ComplEx Minervini et al. (2017)	37.3	41.0	45.9	0.338	**99.2**	**99.3**	**99.4**	**0.984**	71.7	73.6	75.7	0.698
	KBLR Garcia-Duran and Niepert (2018)	–	–	–	–	–	–	–	–	74.0	**77.0**	**79.7**	0.702
Without Rules	TransE Bordes et al. (2013)	36.0	41.5	48.1	0.296	77.5	82.8	88.4	0.630	58.9	64.2	70.2	0.480
	DistMult Yang et al. (2015)	36.0	40.3	45.3	0.313	92.3	93.8	94.7	0.874	67.4	70.2	72.9	0.628
	ComplEx Trouillon et al. (2016)	37.0	41.3	46.2	0.329	91.4	91.9	92.4	0.887	67.3	69.5	71.9	0.641
	GNTPs Minervini et al. (2020)	33.7	36.9	41.2	0.313	**98.2**	**99.0**	**99.3**	**0.977**	69.2	71.1	73.2	0.678
	RotatE Sun et al. (2019b)	**51.1**	**55.1**	**60.3**	**0.471**	86.8	88.6	90.7	0.846	70.8	73.57	77.0	0.678
	CBR **?**	40.0	44.5	48.8	0.359	67.8	71.8	75.9	0.636	57.0	61.2	65.3	0.527
	Our Model	49.0	52.7	57.1	0.457	94.8	95.0	95.3	0.948	**74.2**	**76.0**	**78.2**	**0.727**

Table 3.7: Link prediction results on FB15K-237. Test-II denotes a subset of triples that can be inferred via logical rules.

nearest neighbor and performing local re-arrangements in the form of rotations of tree nodes and global re-arrangments in the form of grafting a subtrees from part of the tree to another. Entities can be deleted from a hierarchy by simply removing the corresponding leaf node. We first use GRINCH to delete the entities whose representations had changed because of the addition of the new node and then incrementally add those entities back along with the newly added entities in the KG. We extract a flat clustering from the hierarchical clustering built by GRINCH using the same method as in §3.6.2.

(c) **Re-estimating new parameters**: After re-assigning clusters, the final step is to estimate the per-cluster parameters. This computation is efficient as it is clear from equations 3.2 and 3.3 that the contribution from each entity in a cluster can be computed independently (and hence can be easily parallelized). However, even for each entity, this computation needs path traversal in the KG which is expensive. We show that we do not have to re-compute for all entities in the clusters.

Let n denote the maximum length of a reasoning path considered by our model. For every new entity e_i added to the KG, we need to recompute statistics for entities that lie within cycles of length up to $(n+1)$ starting from e_i. Please refer to appendix (**??**) for a justification of this result.

3.6.3 Experiments

In this section, we evaluate our proposed approach on a wide array of knowledge-base completion (KBC) benchmarks (§3.6.3). To evaluate the non-parametric nature of our approach, we also evaluate on an 'open-world' setting (§3.6.2) in which new entities are added to the KG. We demonstrate our proposed approach is competitive to several state-of-the-art methods on benchmarks in the standard setting, but it greatly outperforms other methods in the online setting (§3.6.3). The best hyper-parameters for all experiments including the range of hyper-parameter tried and results on validation set are noted in §**??**.

Data and Evaluation Protocol

Data. We evaluate on the following KBC datasets: **NELL-995**, **FB122** Guo et al. (2016), **WN18RR** Dettmers et al. (2018b). **FB122** is a subset of the dataset derived from Freebase, FB15K Bordes et al. (2013), containing 122 relations regarding people, locations, and sports. NELL-995 Xiong et al. (2017) a subset of the NELL

Metric	TransE	DistMult	ComplEx	ConvE	RotatE	GNTP	MINERVA	CBR	Our Model
HITS@1	-	0.39	0.41	0.40	**0.43**	0.41	0.40	0.38	**0.43**
HITS@3	-	0.44	0.46	0.44	**0.49**	0.44	0.43	0.46	**0.49**
HITS@10	0.50	0.49	0.51	0.52	**0.57**	0.48	0.49	0.51	0.55
MRR	0.23	0.43	0.44	0.43	**0.48**	0.43	0.43	0.43	**0.48**
HITS@1	0.53	0.61	0.61	0.67	0.65	-	0.66	0.70	**0.77**
HITS@3	0.79	0.73	0.76	0.81	0.82	-	0.77	0.83	**0.85**
HITS@10	0.87	0.79	0.83	0.86	0.87	-	0.83	0.87	**0.89**
MRR	0.67	0.68	0.69	0.75	0.74	-	0.72	0.77	**0.81**

Table 3.8: Results on WN18RR (above) and NELL-995 (tail-prediction;below)

derived from the 995th iteration of the system. WN18RR was created by Dettmers et al. (2018b) from WN18 by removing inverse relation test-leakage.

Evaluation metrics. Following previous work, we evaluate our method using HITS@N and mean reciprocal rank (MRR), which are standard metrics for evaluating a ranked list.

Experimental Setting

Knowledge Base Completion. Given an entity e_1 and a relation r, our task is retrieve all entities e_2 such that (e_1, r, e_2) belongs in the edges E in a KG $\mathcal{G}$. This task is known as *tail prediction*. If the relation is instead the inverse relation r^{-1}, we assume that we are given an e_2' and asked to predict entities e_1' such that (e_1', r^{-1}, e_2') belongs in the edges E (*head prediction*). To be exactly comparable to baselines, we report an average of head and tail prediction results[2]. We are given a knowledge graph with three partitions of edges, E_{train}, E_{dev}, E_{test}.

For this task, we evaluate against several state-of-the-art embeddings based models such as DistMult Yang et al. (2015), ComplEx Trouillon et al. (2016), ConvE Dettmers et al. (2018b), RotatE Sun et al. (2019b). We also compare against several parametric rule learning methods — NTP Rocktäschel and Riedel (2017b), NeuralLP Yang et al. (2017), MINERVA Das et al. (2018), GNTP Minervini et al. (2020) and also the closely related CBR approach of **?**.

Open-world Knowledge Base Completion. In this setting, we begin with the top 10% of the most popular nodes (with several edges going out from them) and add more randomly selected nodes such that the initial seed KB contains 50% of all the entities in $\mathcal{V}$. This is to ensure, that the seed KB is not too sparse and the initial models trained on them are meaningful. Next, any edges between the nodes selected are added to the seed KB. We divide the rest of the entities randomly into 10 batches. Each batch of entities is incrementally added to the KB along with the edges contained in it. The validation and test set are also divided in the same way, i.e. if both the head and tail entity of a triple are present in the KB, only then the triple is put in the corresponding splits.

Parametric models for KBC that learn representations for a fixed set of entities can not handle 'open-world' setting out-of-the-box. We extend the most competitive embedding based model - RotatE Sun et al. (2019b) for this task. For every new entity arriving in a batch, we initialize a new entity embedding for it. We explore two ways of initializing the new entity embeddings — (a) random initialization, and (b) average of element-wise rotation of entity embeddings w.r.t the relation that this new entity is connected to. Specifically, let t denote the new entity and let $S = \{(h, r, t)\}$ be the facts associated with entity t. Then the embedding $\mathbf{e_t}$ is computed as

$$\mathbf{e_t} = \frac{\sum_{(h,r,t)\in S} \mathbf{e_h} \circ \mathbf{e_r}}{|S|} \tag{3.4}$$

Here, $\circ$ represents the Hadamard (or element-wise) product. This initialization minimizes the RotatE objective for the new embedding ensuring that it is "well-placed" according to the model in the previous

[2]except for NELL-995 dataset where like our baselines, we report tail-prediction performance.

	Our Method	Our Method w/o clustering
HITS@1	0.42	0.29
HITS@3	0.46	0.36
HITS@10	0.51	0.45
MRR	0.45	0.34

Table 3.9: Impact of clustering on WN18RR

	RotatE	Our Method ($n = 3$)	Our Method ($n = 5$)
HITS@1	0.43	0.42	0.43
HITS@3	0.49	0.46	0.49
HITS@10	0.57	0.51	0.55
MRR	0.48	0.45	0.48

Table 3.10: Impact of path length on WN18RR

time step. Embeddings for new relations are initialized randomly. Next, the model is further trained on the new batch of triples so that the new entity embeddings get trained. Note, for massive KGs, it might be impractical to re-train on the entire data as new batches of data arrive frequently, however to still prevent the model to forget what it had learned before, we also sample $m\%$ of triples that it had already been trained on and re-train on them. We ensure that triples in the neighborhood of the newly added entities are ten times likely to be sampled more than other triples. We also try a setting where we try freezing the initially trained entity embeddings and only training the new entity and relation embeddings.

Results on KBC benchmarks

The results for KBC tasks are presented in Table 3.7 and 3.8[3]. Our method does significantly better than parametric rule learning approaches such as MINERVA, GNTPs and the recent case-based approach of **?**. We would like to highlight the difference between the performance of our model and that of **?** on the test-II evaluation of FB122 where triples can be answered by learning logical rules. This results emphasizes the importance of our probabilistic weighing of paths. We also perform comparably to most embedding based models and achieve state-of-the-art results on the overall test sets of FB122 and NELL-995. We report the mean over 3 runs for our model.

We perform an ablation where we do not cluster entities (i.e. every entity has its own cluster) and have per-entity parameters. Table 3.9 notes the drop in performance due to the noisy estimates of path prior and precision parameters because of sparsity. Table 3.11 shows an example where our model learns to score different paths based on the type of entities present in the cluster.

Effect of path length on WN18RR: On the dev set of WN18RR, out of 2985 queries where our method does not rank the answer in the top-10, 2030 queries require a minimum path length greater than 3. Path-based reasoning models have no power to answer these queries. To correct for this, we perform an experiment with the path length $n = 5$ (950 of 2030 answers are reachable). The results in Table 3.10 show that our method recovers a significant portion of performance when allowed to use longer reasoning paths.

Open-World KBC results

Figure 3.5 reports the result for this task. We report results on the RotatE model with randomly initialized embeddings for new entities (RotatE) and the model with systematic initialization of new entity embeddings (RotatE+). We experiment with $m = \{10\%, 30\%\}$ of previously seen edges and re-train on them. We find that not including previously seen edges leads to severe degradation of overall performance due to the model forgetting what it had learned in the past. We also report results with freezing the already seen

[3]There are no reported results of GNTPs on NELL-995

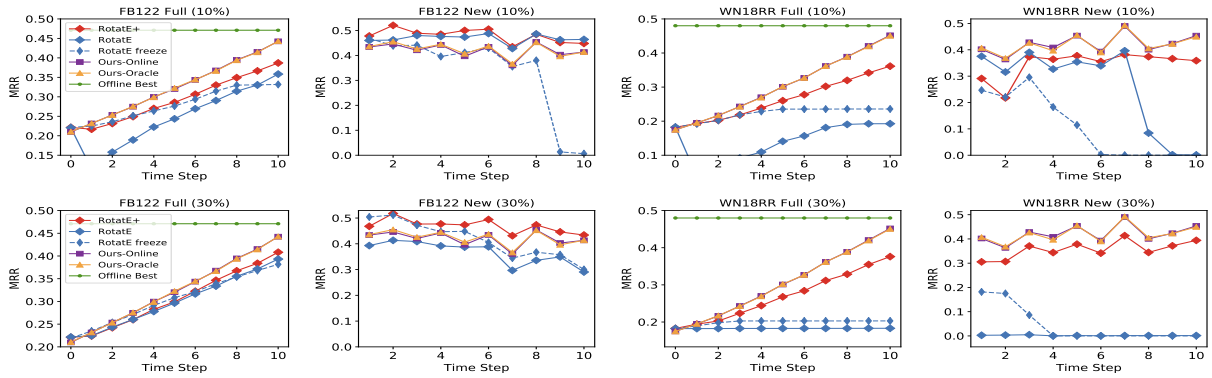


Figure 3.5: Results for open-world setting when trained with 10% (top row) and 30% (bottom row) of already seen edges. Our online method matches the offline version of our approach and outperforms the online variants of RotatE. After all data is observed our online method achieves results closest to the best offline method's results.

Athlete Cluster	(athlete-led-sports-team, team-plays-in-league) (athlete-home-stadium, league-stadiums^{-1})
Politician Cluster	(politician-us-member-of-political-group, person-belongs-to-organization^{-1}, agent-belongs-to-organization) (agent-collaborates-with-agent, agent-belongs-to-organization)

Table 3.11: High scoring paths in different clusters for the query agent-belongs-to-organization in NELL-995

entity representations and only learning representations for new entities (RotatE-Freeze). All models were trained till the validation set (containing both new and old triples) performance stopped improving. For our approach, we also report results for an oracle setting where we re-cluster all entities as new data arrives and re-estimate all parameters from scratch (instead of using GRINCH and recomputing only required parameters (§3.6.2). For both datasets, the offline-best results were obtained by RotatE (47.1 for FB122 test-I, 48 for WN18RR). We report performance on the entire evaluation set (full) and also on the set containing the newly added edges (new).

The main summary of the results are (i) RotatE model converges to a much lower performance in the online setting losing at least 8 MRR points in FB122 and at least 11 points in WN18RR. On FB122, we observe that the model prefers to learn new information more by sacrificing previously learned facts (2nd subfigure in figure 3.5) (ii) In the freeze setting, the model performance deteriorates quickly after a certain point indicating saturation, i.e. it becomes hard for the model to learn new information about arriving entities by keeping the parameters of the existing entities fixed. (iii) On the full evaluation, RotatE+ performs better than RotatE showing that bad initialization deteriorates performance over time, however, there is still a large gap between the best performance (iv) Our approach almost matches our performance in oracle setting indicating the effectiveness of the online clustering and fast parameter approximation. (v) Lastly, we perform closest to the offline best results outperforming all variants of RotatE.

3.6.4 Related Work

Open-world KG completion. Shi and Weninger (2018) consider the task of open-world KG completion. However, they use text descriptions to learn entity representations using convolutional neural networks. Our model does not use additional text data and we use very simple entity representations that helps us to perform well. Tang et al. (2019) learns to update a KG with new links by reading news. Even though they handle adding or deleting new edges, they do not observe new entities. Lastly, none of them learn from similar entities using a CBR approach.

Inductive representation learning on KGs. Recent works Teru et al. (2020b); Wang et al. (2020) learn entity independent relation representations and hence allow them to handle unseen entities. However, they do not perform contextual reasoning by gathering reasoning paths from similar entities. Moreoever, in our open-world setting, we consider the more challenging setting, where new facts and entities are arriving in a streaming fashion and we give an efficient way of updating parameters using online hierarchical clustering. This allows our method to be applicable in settings where the initial KG is small and it grows continuously.

Rule induction in knowledge graphs. Classic work in inductive logic programming (ILP) Muggleton et al. (1992); Quinlan (1990) induce rules from grounded facts. However, they need explicit counter-examples which are not present in KBs and they do not scale to large KBs. Recent ILP approaches Galárraga et al. (2013, 2015) try to fix this deficiency by guessing counter examples from rules and making it more scalable. Statistical relational learning methods Getoor and Taskar (2007); Kok and Domingos (2007); Schoenmackers et al. (2010) and probabilistic logic approaches Richardson and Domingos (2006); Broecheler et al. (2010); Wang et al. (2013) combine machine learning and logic to learn rules. However, none of these work derive reasoning rules dynamically from similar entities in the knowledge graph.

Bayesian non-parametric approaches for link-prediction. There is a rich body of work in bayesian non-parametrics to automatically learn the latent dimension of entities Kemp et al. (2006); Xu et al. (2006). Our method does not learn latent dimension of entities, instead our work is non-parametric because it gathers reasoning paths from nearest neighbors and can seamlessly reason with new entities by efficiently updating parameters using online non-parametric hierarchical clustering.

Embedding-based approach for link prediction. We also compare to the more popular embeddings based models based on tensor factorization or neural approaches Nickel et al. (2011b); Bordes et al. (2013); Dettmers et al. (2018b); Sun et al. (2019b). Our simple approach which needs no iterative optimization outperforms most of them and performs comparably to the latest RotatE model. Moreover we outperform RotatE in the online experiments.

CBR for KG completion. There has been few attempts to apply CBR for knowledge management Dubitzky et al. (1999); Bartlmae and Riemenschneider (2000), however they do not do contextualized reasoning or consider online settings. Our work is most closely related to the recent work of **?**. However, since it does not take in to account the importance of each path, it suffers from low performance, with our model outperforming it in several benchmarks.

3.6.5 Conclusion

We present a simple yet accurate approach for probabilistic case-based reasoning in knowledge bases. Our method is non-parametric, deriving reasoning rules dynamically from similar entities in the KB and is capable of handling new entities. We cluster similar entities together and estimate per-cluster parameters that measures the prior and precision of paths using simple count statistics. Our simple approach performs competitively to the best embeddings based models on several benchmarks and outperforms all models in the open-world setting.

4 Case-Based Reasoning for Natural Language Queries over Knowledge Bases

4.1 Introduction

Humans often solve a new problem by recollecting and adapting the solution to multiple related problems that they have encountered in the past Ross (1984); Lancaster and Kolodner (1987); Schmidt et al. (1990). In classical artificial intelligence (AI), case-based reasoning (CBR) pioneered by Schank (1982), tries to incorporate such model of reasoning in AI systems Kolodner (1983); Rissland (1983); Leake (1996). A sketch of a CBR system Aamodt and Plaza (1994) comprises of — (i) a retrieval module, in which 'cases' that are similar to the given problem are retrieved, (ii) a reuse module, where the solutions of the retrieved cases are re-used to synthesize a new solution. Often, the new solution does not work and needs more revision, which is handled by a (iii) revise module.

In its early days, the components of CBR were implemented with symbolic systems, which had their limitations. For example, finding similar cases and synthesizing new solutions from them is a challenging task for a CBR system implemented with symbolic components. However, with the recent advancements in representation learning LeCun et al. (2015), the performance of ML systems have improved substantially on a range of practical tasks.

Given a query, CBR-KBQA uses a neural retriever to retrieve other similar queries (and their logical forms) from a case memory (e.g. training set). Next, CBR-KBQA generates a logical form for the given query by learning to reuse various components of the logical forms of the retrieved cases. However, often the generated logical form does not produce the right answer when executed against a knowledge base (KB). This can happen because one or more KB relations needed are never present in the retrieved cases or because KBs are woefully incomplete Min et al. (2013) (Figure 4.1). To alleviate such cases, CBR-KBQA has an additional revise step that *aligns* the generated relations in the logical form to the query entities' local neighborhood in the KB. To achieve this, we take advantage of pre-trained relation embeddings from KB completion techniques (e.g. Trans-E Bordes et al. (2013)) that learn the structure of the KB.

It has been shown that neural seq2seq models do not generalize well to novel compositions of previously seen input Lake and Baroni (2018); Loula et al. (2018). However, CBR-KBQA has the ability to reuse relations from *multiple* retrieved cases, even if each case contains only partial logic to answer the query. We show that CBR-KBQA is effective for questions that need novel compositions of KB relations, achieving state-of-the-art results on multiple KBQA benchmarks such as WebQuestionsSP Yih et al. (2016), ComplexWebQuestions (CWQ) Talmor and Berant (2018) and CompositionalFreebaseQuestions (CFQ) Keysers et al. (2020). For example, on the hidden test-set of the challenging CWQ dataset, CBR-KBQA outperforms the best system by over 11% points.

We further demonstrate that CBR-KBQA, without the need of any further fine-tuning, also generalizes to queries that need relations that were *never seen* in the training set. This is possible due to CBR-KBQA's nonparametric approach which allows one to *inject* relevant simple cases during inference, allowing it to reuse new relations from those cases. In a controlled human-in-the-loop experiment, we show that CBR-KBQA can correctly answer such questions when an expert (e.g. database administrator) injects few simple cases to the case memory. CBR-KBQA is able to retrieve those examples from the memory and use the unseen relations to compose new logical forms for the given query.

Generalization to unseen KB relations, without any re-training, is out of scope for current neural models. Currently, the popular approach to handle such cases is to re-train or fine-tune the model on new examples. This process is not only time-consuming and laborious but models also suffer from catastrophic forgetting Hinton and Plaut (1987); Kirkpatrick et al. (2017), making wrong predictions on examples which it previously predicted correctly. We believe that the controllable properties of CBR-KBQA are essential for QA models to be deployed in real-world settings and hope that our work will inspire further research in this direction.

Recent works such as REALM Guu et al. (2020) and RAG Lewis et al. (2020b) retrieve relevant paragraphs

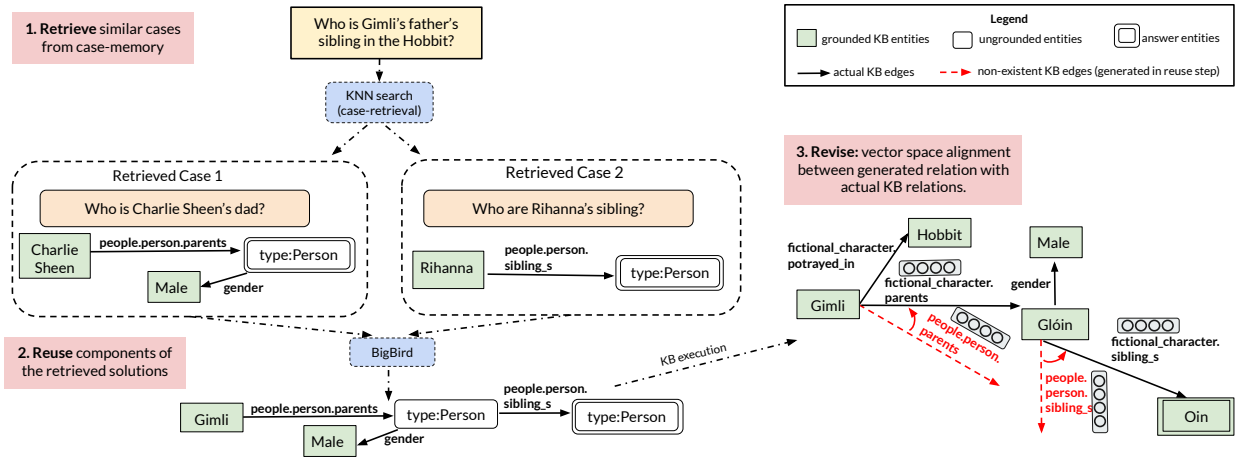


Figure 4.1: CBR-KBQA derives the logical form (LF) for a new query from the LFs of other retrieved queries from the case-memory. However, the derived LF might not execute because of missing edges in the KB. The revise step aligns any such missing edges (relations) with existing semantically-similar edges in the KB.

from a nonparametric memory for answering questions. CBR-KBQA, in contrast, retrieves *similar queries* w.r.t the input query and uses the relational similarity between their logical forms to derive a logical form for the new query. CBR-KBQA is also similar to the recent retrieve and edit framework Hashimoto et al. (2018) for generating structured output. However, unlike us they condition on only a single retrieved example and hence is unlikely to be able to handle compositional questions. Moreover, unlike CBR-KBQA, retrieve and edit does not have a component that can explicitly revise an initially generated output.

The contributions of our paper are as follows — (a) We present a neural CBR approach for KBQA capable of generating complex logical forms conditioned on similar retrieved questions and their logical forms. (b) Since CBR-KBQA explicitly learns to reuse cases, we show it is able to generalize to unseen relations at test time, when relevant cases are provided. (c) We also show the efficacy of our revise step of CBR-KBQA which allows to correct generated output by aligning it to local neighborhood of the query entity. (d) Lastly, we show that CBR-KBQA significantly outperforms other competitive models on several KBQA benchmarks.

4.2 Model

This section describes the implementation of various modules of CBR-KBQA. In CBR, a case is defined as an abstract representation of a problem along with its solution. In our KBQA setting, a case is a natural language query paired with an executable logical form. The practical importance of KBQA has led to the creation of an array of recent datasets (Zelle and Mooney, 1996; Bordes et al., 2015; Su et al., 2016; Yih et al., 2016; Zhong et al., 2017; Ngomo, 2018; Yu et al., 2018; Talmor and Berant, 2018, inter-alia). In these datasets, a question is paired with an executable logical form such as SPARQL, SQL, S-expression or graph query. All of these forms have equal representational capacity and are interchangeable Su et al. (2016). Figure 4.2 shows an example of two equivalent logical forms. For our experiments, we consider SPARQL programs as our logical form.

Formal definition of task: let q be a natural language query and let $\mathcal{K}$ be a symbolic KB that needs to be queried to retrieve an answer list $\mathcal{A}$ containing the answer(s) for q. We also assume access to a training set $\mathcal{D} = \{(q_1, \ell_1), (q_2, \ell_2), \ldots (q_N, \ell_N)\}$ of queries and their corresponding logical forms where q_i, ℓ_i represents query and its corresponding logical form, respectively. A logical form is an executable query containing entities, relations and free variables (Figure 4.2). CBR-KBQA first retrieves K similar cases $\mathcal{D}_q$ from $\mathcal{D}$ (§ 4.2.1). It then generates a intermediate logical form ℓ_{inter} by learning to reuse components of the logical forms of the retrieved cases (§ 4.2.2). Next, the logical form ℓ_{inter} is revised to output the final logical form ℓ by aligning to the relations present in the neighborhood subgraph of the query entity to recover from any spurious

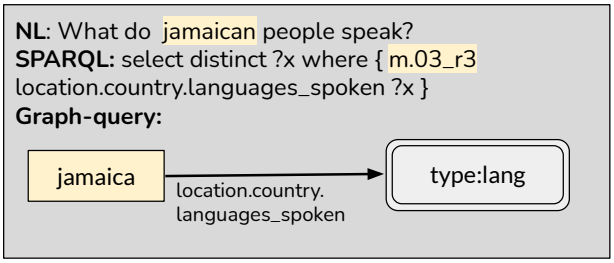


Figure 4.2: An example of a SPARQL logical form for a simple query and its equivalent graph-query.

relations generated in the reuse step (§ 4.2.3). Finally, ℓ is executed against $\mathcal{K}$ and the list of answer entities are returned. We evaluate our KBQA system by calculating the accuracy of the retrieved answer list w.r.t a held-out set of queries.

4.2.1 Retrieve

The retrieval module computes dense representation of the given query and uses it to retrieve other similar query representation from a training set. Inspired by the recent advances in neural dense passage retrieval Das et al. (2019); Karpukhin et al. (2020), we use a ROBERTA-base encoder to encode each question independently. Also, we want to retrieve questions that have high relational similarity instead of questions which share the same entities (e.g. we prefer to score the query pair (Who is Justin Bieber's brother?, Who is Rihanna's brother?), higher than (Who is Justin Bieber's brother?, Who is Justin Bieber's father?)). To minimize the effect of entities during retrieval, we use a named entity tagger[1] to detect spans of entities and mask them with [BLANK] symbol with a probability p_{mask}, during training. The entity masking strategy has previously been successfully used in learning entity-independent relational representations Soares et al. (2019). The similarity score between two queries is given by the inner product between their normalized vector representations (cosine similarity), where each representation, following standard practice Guu et al. (2020), is obtained from the encoding of the initial [CLS] token of the query.

Fine-tuning question retriever: In passage retrieval, training data is gathered via distant supervision in which passages containing the answer is marked as a positive example for training. Since in our setup, we need to retrieve similar questions, we use the available logical forms as a source of distant supervision. Specifically, a question pair is weighed by the amount of overlap (w.r.t KB relations) it has in their corresponding logical queries. Following DPR Karpukhin et al. (2020), we ensure there is atleast one positive example for each query during training and use a weighted negative log-likelihood loss where the weights are computed by the F_1 score between the set of relations present in the corresponding logical forms. Concretely, let $(q_1, q_2, \ldots, q_B)$ denote all questions in a mini-batch. The loss function is:

$$L = -\sum_{i,j} w_{i,j} \log \frac{\exp(\text{sim}(\mathbf{q_i}, \mathbf{q_j}))}{\sum_j \exp(\text{sim}(\mathbf{q_i}, \mathbf{q_j}))}$$

Here, $\mathbf{q_i} \in \mathbb{R}^d$ denotes the vector representation of query q_i and $\text{sim}(\mathbf{q_i}, \mathbf{q_j}) = \mathbf{q_i}^\top \mathbf{q_j}$. $w_{i,j}$ is computed as the F_1 overlap between relations in the logical pairs of q_i and q_j. We pre-compute and cache the query representations of the training set $\mathcal{D}$. For query q, we return the top-k similar queries in $\mathcal{D}$ w.r.t q and pass it to the reuse module.

4.2.2 Reuse

The reuse step generates an intermediate logical form from the k cases that are fed to it as input from the retriever module. Pre-trained encoder-decoder transformer models such as BART Lewis et al. (2020a) and T5 Raffel et al. (2020) have enjoyed dramatic success on semantic parsing Lin et al. (2018b); Hwang et al. (2019);

[1] https://cloud.google.com/natural-language

Shaw et al. (2020); Suhr et al. (2020). We take a similar approach in generating an intermediate logical form conditioned on the retrieved cases. However, one of the core limitation of transformer-based models is its quadratic dependency (in terms of memory), because of full-attention, which severely limits the sequence length it can operate on. For example, BART and T5 only supports sequence length of 512 tokens in its encoder. Recall that for us, a case is a query from the train set and an executable SPARQL program, which can be arbitrarily long.

To increase the number of input cases, we leverage a recently proposed sparse-attention transformer architecture — BIGBIRD Zaheer et al. (2020). Instead of having each token attend to all input tokens as in a standard transformer, each token attends to only nearby tokens. Additionally, a small set of global tokens attend to all tokens in the input. This reduces the transformer's memory complexity from quadratic to linear, and empirically, BIGBIRD enables us to use many more cases.

Description of input: The input query q and cases $\mathcal{D}_q = \{(q'_1, \ell'_1), (q'_2, \ell'_2), \ldots (q'_k, \ell'_k)\}$ are concatenated on the encoder side. Specifically, $\texttt{Input}_{\texttt{ENC}}(\texttt{q}, \mathcal{D}_\texttt{q}) = \texttt{q [SEP]} \texttt{q}'_1 \texttt{ [SEP]} \ell'_1, \ldots \texttt{q}_\texttt{k}' \texttt{ [SEP]} \ell_\texttt{k}'$, where `[SEP]` denotes the standard separator token. Each logical form also contain the KB entity id of each entities in the question (e.g. m.03_r3 for Jamaica in Figure 4.2). We append the entity id after the surface form of the entity mention in the question string. For example, the query in Figure 4.2 becomes "What do Jamaican m.03_r3 people speak?".

Training is done using a standard seq2seq cross-entropy objective. Large deep neural networks usually benefit from "good" initialization points Frankle and Carbin (2019) and being able to utilize pre-trained weights is critical for seq2seq models. We find it helpful to have a regularization term that minimizes the Kullback–Leibler divergence (KLD) between output softmax layers of (1) when only the query q is presented (i.e not using cases), and (2) when query and cases ($\mathcal{D}_q$) are available Yu et al. (2013). Formally, let f be the seq2seq model, let $\sigma = softmax(f(q, \mathcal{D}_q))$ and $\sigma' = softmax(f(q))$ be the decoder's prediction distribution with and without cases, respectively. The following KLD term is added to the seq2seq cross-entropy loss

$$L = L_{CE}(f(q, \mathcal{D}_q), l_q) + \lambda_T \text{KLD}(\sigma, \sigma')$$

where $\lambda_T \in [0, 1]$ is a hyper-parameter. Intuitively, this term regularizes the prediction of $f(q, \mathcal{D}_q)$ not to deviate too far away from that of the $f(q)$ and we found this to work better than initializing with a model not using cases.

4.2.3 Revise

In the previous step, the model explicitly reuses the relations present in $\mathcal{D}_q$, nonetheless, there is no guarantee that the query relations in $\mathcal{D}_q$ will contain the relations required to answer the original query q. This can happen when the domain of q and domain of cases in $\mathcal{D}_q$ are different even when the relations are semantically similar. For example, in Figure 4.1 although the retrieved relations in NN queries are semantically similar, there is a domain mismatch (person v/s fictional characters). Similarly, large KBs are very incomplete Min et al. (2013), so querying with a valid relation might require an edge that is missing in the KB leading to intermediate logical forms which do not execute.

To alleviate this problem and to make the queries executable, we explicitly *align* the generated relations with relations (edges) present in the local neighborhood of the query entity in the KG. We propose the following alignment models:

Using pre-trained KB embeddings: KB completion is a extensively studied research field Nickel et al. (2011b); Bordes et al. (2013); Socher et al. (2013); Velickovic et al. (2018); Sun et al. (2019b) and several methods have been developed that learn low dimensional representation of relations such that similar relations are closer to each other in the embedding space. We take advantage of the pre-trained relations obtained from TransE Bordes et al. (2013), a widely used model for KB completion. For each predicted relation outgoing from / incoming to an entity, we find the most similar relation edge (in terms of cosine similarity) that exists in the KB for that entity and align with it. If the predicted edge exists in the KB, it trivially aligns with itself. There can be multiple missing edges that needs aligning (Figure 4.1) and we find it more effective to do beam-search instead of greedy-matching the most similar edge at each step.

Model	P	R	F1	Acc
STAGG Yih et al. (2016)	70.9	**80.3**	71.7	63.9
GraftNet Sun et al. (2018)	-	-	66.4†	-
PullNet Sun et al. (2019a)	-	-	68.1†	-
EmbedKGQA Saxena et al. (2020)	-	-	66.6†	-
T5-11B Raffel et al. (2020)	62.1	62.6	61.5	56.5
T5-11B + Revise	63.6	64.3	63.0	57.7
CBR-KBQA (Ours)	**73.1**	75.1	**72.8**	**70.0**

Table 4.1: Performance on the WebQSP dataset. GraftNet, PullNet and EmbedKGQA produces a ranking of KG entities hence evaluation is in Hits@k (see text for description). CBR-KBQA significantly outperforms baseline models in the strict exact match accuracy metric. † Models report hits@1 instead of F1

Using similarity in surface forms: Similar relations (even across domains) have overlap in their surface forms (e.g. 'siblings' is common term in both 'person.siblings' and 'fictional_character.siblings'). Therefore, word embeddings obtained by encoding these words should be similar. This observation has been successfully utilized in previous works Toutanova and Chen (2015); Hwang et al. (2019). We similarly encode the predicted relation and all the outgoing or incoming edges with ROBERTA-base model. Following standard practices, relation strings are prepended with a [CLS] token and the word pieces are encoded with the ROBERTA-base model and the output embedding of the [CLS] token is considered as the relation representation. Similarity between two relation representations is computed by cosine similarity.

Our alignment is simple and requires no learning. By aligning only to individual edges in the KB, we make sure that we do not change the structure of the generated LF. We leave the exploration of learning to align single edges in the program to sequence of edges (paths) in the KB as future work.

4.3 Experiments

Data: For all our experiments, the underlying KB is full Freebase containing over 45 million entities (nodes) and 3 billion facts (edges) Bollacker et al. (2008). We test CBR-KBQA on three datasets — WebQSP Yih et al. (2016), CWQ Talmor and Berant (2018) and CFQ Keysers et al. (2020). Please refer to §**??** for a detailed description of each datasets.

Hyperparameters: All hyperparameters are set by tuning on the valdation set for each dataset. We initialize our retriever with the pre-trained ROBERTA-base weights. We set $p_{\text{mask}} = 0.2$ for CWQ and 0.5 for the remaining datasets. We use a BIGBIRD generator network with 6 encoding and 6 decoding sparse-attention layers, which we initialize with pre-trained BART-base weights. We use k=20 cases and decode with a beam size of 5. Initial learning rate is set to 5×10^{-5} and is decayed linearly through training. Further details for the EMNLP reproducibility checklist is given in §**??**.

4.3.1 Entity Linking

The first step required to generate an executable LF for a NL query is to identify and link the entities present in the query. For our experiments, we use a combination of an off-the-shelf entity linker and a large mapping of mentions to surface forms. For the off-the-shelf linker, we use a recently proposed high precision entity linker ELQ Li et al. (2020). To further improve recall of our system, we first identify mention spans of entities in the question by tagging it with a NER[2] system. Next, we link entities not linked by ELQ by exact matching with surface form annotated in FACC1 project Gabrilovich et al. (2013). Our entity linking results are shown in Table 4.2.

[2] https://cloud.google.com/natural-language

Dataset	Precision	Recall	F1
WebQSP	0.761	0.819	0.789
CWQ	0.707	0.910	0.796

Table 4.2: Entity linking performance on various datasets

Model	P	R	F1	Acc
KBQA-GST Lan et al. (2019)	-	-	-	39.4
QGG Lan and Jiang (2020)	-	-	-	44.1
PullNet Sun et al. (2019a)	-	-	-	45.9
DynAS (Anonymous)	-	-	-	50.0
T5-11B Raffel et al. (2020)	55.2	55.4	54.6	52.4
T5-11B + Revise	58.7	59.6	58.2	55.6
CBR-KBQA (Ours)	**70.4**	**71.9**	**70.0**	**67.1**

Table 4.3: Performance on the hidden test set of CWQ.

4.3.2 KBQA Results

Table 4.1 reports results on WebQSP. All reported model except CBR-KBQA and T5-11B directly operate on the KB (e.g. traverse KB paths starting from the query entity) to generate the LF or the answer. As a result, models such as STAGG tend to enjoy much higher recall. On the other hand, much of our logical query is generated by reusing components of similar cases. Models such as GraftNet Sun et al. (2018) and PullNet Sun et al. (2019a) rank answer entities and return the top entity as answer (Hits@1 in table 4.2). This is undesirable for questions that have multiple entities as answers (e.g. "Name all major cities in the U.S.?"). We also compare to a large pre-trained seq2seq model with over 11B parameters — T5 Raffel et al. (2020), which was recently shown to be effective for compositional KBQA Furrer et al. (2020). T5 was fine-tuned on the query and LF pair. We also report the results of 'aligning' the LF produced by T5 using our revise step. As shown in Table 4.1, CBR-KBQA outperforms all other models significantly and improves on the strict exact-match accuracy by more than 6 points w.r.t. the best model. Revise step improves on the performance of T5 suggesting that it is generally applicable. Table 4.3 reports performance on the hidden test set of CWQ, which was built by extending WebQSP questions with the goal of making a more complex multi-hop questions. It is encouraging to see that CBR-KBQA outperforms all other baselines on this challenging dataset by a significant margin. Finally, we report results on CFQ in Table 4.4. The creators of CFQ propose to evaluate performance with exact string match accuracy between SPARQL programs is computed, which is quite conservative in what is counted as correct. For example, in a SPARQL query having a different (but consistent) name for a free variable (e.g. x1 instead of x) does not change the semantics of the query. Also SPARQL is independent of the order of the relational triples. This is especially unfair to our model, which copies relations from other nearest neighbor queries and can copy relations in any order. As a remedy, we report results by *executing* the predicted (and gold) queries against a Freebase KB, instead of string-match performance. We also recompute the results of the T5-large model (which also improves significantly (40.9 v/s 67.7) w.r.t. the original results reported in Furrer et al. (2020)). CBR-KBQA outperforms the baseline on this dataset as well.

4.3.3 Efficacy of Revise step

Table 4.5 show that the revise step is useful for CBR-KBQA on multiple datasets. We also show that the T5 model also benefits from the alignment in revise step with more than 3 points improvement in F1 score on the CWQ dataset. We find that TransE alignment outperforms ROBERTA based alignment, suggesting that graph structure information is more useful than surface form similarity for aligning relations. Moreover, relation names are usually short strings, so they do not provide enough context for LMs to form good representations.

Next we demonstrate the advantage of the nonparametric property of CBR-KBQA— ability to fix an initial wrong prediction by allowing new cases to be *injected* to the case-memory. This allows CBR-KBQA to

Model	MCD1	MCD2	MCD3	MCD-mean
T5-11B	72.9	69.2	62.0	67.7
CBR-KBQA	**87.8**	**75.1**	**71.5**	**78.1**

Table 4.4: Performance (accuracy) on the CFQ dataset.

WebQSP	Accuracy(%)	Δ
CBR-KBQA (before Revise)	69.43	–
+Revise (Roberta)	69.49	+0.06
+Revise (TransE)	**70.00**	**+0.57**
CWQ	Accuracy(%)	Δ
CBR-KBQA (before Revise)	65.95	–
+Revise (Roberta)	66.32	+0.37
+Revise (TransE)	**67.11**	**+1.16**

Table 4.5: Impacts of the revise step. We show that the revise step consistently improves the accuracy on WebQSP and CWQ, especially with the TransE pretrained embeddings.

generalize to queries which needs relation *never seen* during training. Due to space constraints, we report other results (e.g. retriever performance), ablations and other analysis in §**??**.

4.3.4 Point-Fixes to Model Predictions

Modern QA systems built on top of large LMs do not provide us the opportunity to *fix* an erroneous prediction. The current approach is to fine-tune the model on new data. However, this process is time-consuming and impractical for production settings. Moreover, it has been shown (and as we will empirically demonstrate) that this approach leads to catastrophic forgetting where the model forgets what it had learned before. McCloskey and Cohen (1989); Kirkpatrick et al. (2017). On the other hand, CBR-KBQA adopts a nonparametric approach and allows inspection of the retrieved nearest neighbors for a query. Moreover, one could *inject* a new relevant case into the case-memory (KNN-index), which could be picked up the retriever and used by the reuse module to fix an erroneous prediction.

Performance on Unseen Relations

We consider the case when the model generates a wrong LF for a given query. We create a controlled setup by removing all queries from the training set of WebQSP which contain the (people.person.education) relation. This led to a removal of 136 queries from the train set and ensured that the model failed to correctly answer the 86 queries (held-out) in the test set which contained the removed relation in its LF.

We compare to a baseline transformer model (which do not use cases) as our baseline. As shown in Table 4.6, both baseline and CBR-KBQA do not perform well without any relevant cases since a required KB relation

Scenario	Initial Set	Held-Out
Transformer	59.6	0.0
+ Fine-tune on additional cases only (100 gradient steps)	1.3	76.3
+ Fine-tune on additional cases and original data (300 gradient steps)	53.1	57.6
CBR-KBQA (Ours)	69.4	0.0
+ Adding additional cases to index (0 gradient steps; 2 sec)	69.4	70.6

Table 4.6: Robustness and controllability of our method against black-box transformers. CBR-KBQA can easily and quickly adopt to new relations given cases about it, whereas heavily parameterized transformer is not stable, and can undergo catastrophic forgetting when we try to add new relation information intro its parameters.

Scenario	P	R	F1	Acc
CBR-KBQA (Ours)	0.0	0.0	0.0	0.0
+ additional cases	36.54	38.59	36.39	32.89

Table 4.7: Results for H-I-T-L experiment. After adding a few cases, we see that we can get the accuracy of OOV questions to improve considerably without needing to re-train the model.

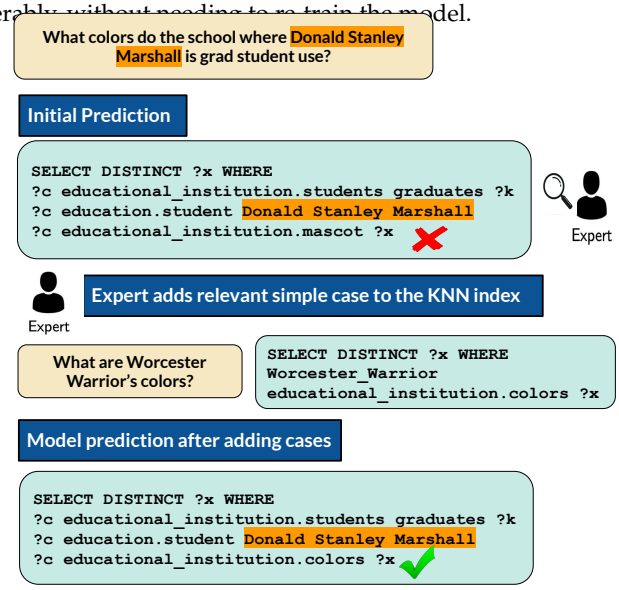


Figure 4.3: An expert point-fixes a model prediction by adding a simple case to the KNN index. Initial prediction was incorrect as no query with the relation (educational_institution.colors) was present in the train set. CBR-KBQA retrieves the case from the KNN index and fixes the erroneous prediction without requiring any re-training.

was missing during training. Next, we add the 136 training instances back to the training set and recompute the KNN index. This process involves encoding the newly added NL queries and recomputing the KNN index, a process which is computationally much cheaper than re-training the model again. Row 5 in Table 4.6 shows the new result. On addition of the new cases, CBR-KBQA can seamlessly use them and copy the unseen relation to predict the correct LF, reaching 70.6% accuracy on the 86 held-out queries.

In contrast, the baseline transformer must be fine-tuned on the new cases to handle the new relation, which is more computationally expensive than adding the cases to our index. Moreover, just fine-tuning on the new instances leads to *catastrophic forgetting* as seen in row 2 of Table 4.6 where the baseline model's performance on the initial set decreases drastically. We find it necessary to carefully fine-tune the model on new examples alongside original training examples (in a 1:2 proportion). However, it still converges to a performance which is lower than its original performance and much lower than the performance of CBR-KBQA.

Human-in-the-Loop Experiment

During error analysis, we realized that there are queries in the test set of WebQSP that contain KB relations in their LFs which were never seen during training[3]. That means model will never be able to predict the correct LF for the query because of the unseen relation. We conduct a human-in-the-loop experiment (Figure 4.3) in which users add *simple* 'cases' to point-fix erroneous predictions of CBR-KBQA for those queries. A simple case is a NL query paired with a program which only contain one KB relation. Table **??** (Appendix **??**) shows some example of such cases. Because of the simple nature of the questions, these cases can be created manually (by a user who is knowledgeable about the KB schema) or automatically curated from data sources such as SimpleQuestions Bordes et al. (2015) which is a large collection of NL queries that can be a mapped to a single KB edge. Table **??** in Appendix **??** shows various statistics of the missing relations and the number of cases added by humans and from SimpleQuestions. The cases are added to the original KNN-index. By adding a few cases, the performance increases from 0 to 36 F1 (Table 4.7) without requiring any training.

[3]There are 94 different unseen relations in test set.

Data	# Total Q	# Q that need comp. reasoning	# Correct	
			T5	CBR
CWQ	3531	639	205	**270**
CFQ	11968	6541	3351	**3886**

Table 4.8: Analysis of Compositional Reasoning. We compare the performance of models on questions that need *novel combinations* of relations *not seen* during training.

Note unlike the previous controlled experiment in §4.3.4, we add around 3.87 cases for each unseen relation[4].
Importance of this result: We believe that flexibility of models to *fix* predictions (without training) is an important desiderata for QA systems deployed in production settings and we hope our results will inspire future research in this direction.

4.3.5 Performance on Compositional Questions

We analyze questions in the evaluation set which require *novel combinations* of relations *never seen* in the training set. This means, in order for our model to answer these questions correctly, it would have to retrieve relevant nearest neighbor (NN) questions from the training set and copy the required relations from the logical form of *multiple* NN queries. Table 4.8 shows that our model outperforms the competitive T5 baseline. Also as we saw in the last section, our model is able to quickly adapt to relations *never seen* in the training set altogether by picking them up from newly added cases.

4.4 Related Work

Retrieval augmented QA models such as, Chen et al. (2017); Guu et al. (2020); Lewis et al. (2020b) augments a reader model with a retriever to find relevant paragraphs from a corpus. In contrast, our CBR approach retrieves *similar queries*, instead of supporting context and uses their solution to derive a new solution. Recently, Lewis et al. (2020c) reported that 60-70% test time answers are usually present in training set and they propose a model that finds a nearest neighbor (NN) question and returns their answer. On the contrary we do not rely on answers but full SPARQL LFs which are unlikely to repeat and we explicitly show (Table 4.8) that CBR-KBQA can derive LFs from multiple related queries. Moreover, instead of returning answers to NN queries, we *learn* to derive a LF for the new query.
Retrieve and edit: CBR-KBQA shares similarities with the RETRIEVE-AND-EDIT framework Hashimoto et al. (2018) which utilizes the case of the nearest-neighbor w.r.t input. They simply rely on a seq2seq model to generate answer from the retrieved case without any revision. Furthermore, our reuse step brings in new challenges as parametric model have to compose one SPARQL query from multiple cases in contrast to them that only considers a single case.
K-NN approach in other NLP applications: Wiseman and Stratos (2019) achieved accurate sequence labeling by explicitly and only copying labels from retrieved neighbors. Khandelwal et al. (2020) also observed improvements in language modeling by utilizing explicit examples from training data. However, unlike us, these work do not necessarily learn to compose from multiple retrieved questions. There has been work in machine translation Zhang et al. (2018a); Gu et al. (2018); Khandelwal et al. (2021) that uses nearest neighbor translation pair to guide the decoding process. In contrast, CBR-KBQA needs to produce logical forms and has the revise step which ensures that the queries can be executed against a KB. Recently, Hossain et al. (2020) proposed a retrieve-edit-rerank approach for text generation in which each retrieved candidate from the training set is edited independently and then re-ranked. In contrast, CBR-KBQA generates the program *jointly* from all the retrieved cases and is more suitable for questions which needs copying relations from multiple nearest neighbors. Please refer to (§**??**) for further related work.

4.5 Limitations and Future Work

To the best of our knowledge, we are the first to propose a neuralized CBR approach for KBQA. We showed that our model is effective in handling complex compositional questions, but our work also has several

[4]In §4.3.4, we added 136 cases (v/s 3.87) for one relation. This is why the accuracy in Table 4.6 is higher w.r.t Table 4.7.

limitations. First, our model relies on the availability of supervised logical forms such as SPARQL queries, which can be expensive to annotate at scale. In the future, we plan to explore ways to directly learn from question-answer pairs Berant et al. (2013); Liang et al. (2016). Even though, CBR-KBQA is modular and has several advantages, the retrieve and reuse components of our model are trained separately. In future, we plan to explore avenues for end to end learning for CBR.

5 Knowledge Base Question Answering by Case-based Reasoning over Subgraphs

5.1 Introduction

Knowledge bases (KBs) store massive amounts of rich symbolic facts about real-world entities in the form of relation triples — $(\mathrm{e}_1, \mathrm{r}, \mathrm{e}_2)$, where e_1, e_2 denote entities and r denotes a semantic relation. KBs can be naturally described as a graph where the entities are nodes and the relations are labelled edges. An effective and user-friendly way of accessing the information stored in a KB is by issuing queries to it. Such queries can be structured (e.g. queries for booking flights) or unstructured (e.g. natural language queries). The set of KB facts useful for answering a query induce a reasoning pattern — e.g. a chain of KB facts forming a path or more generally a subgraph in the knowledge graph (KG) (set of red edges in Figure 5.1). It is very laborious to annotate the reasoning patterns for each query at scale and hence it is important to develop weakly-supervised knowledge base question answering (KBQA) models that do not depend on the availability of the annotated reasoning patterns.

We are interested in developing models that can answer queries that require complex subgraph reasoning patterns. Many previous work in KBQA Neelakantan et al. (2015b); Xiong et al. (2017); He et al. (2021) reason over relational paths in a KB. However, many queries require a model to reason over a set of facts *jointly*. For example, the query in Figure5.1 cannot be answered by considering individual paths. Because of the diverse nature of possible questions, the number of different reasoning patterns that a model has to learn is massive. Moreover a model may encounter very few examples of a particular pattern during training, making it challenging for the models to learn and encode the patterns entirely in its parameters.

Case-based Reasoning (CBR), a semiparametric framework, proposed decades ago in classical AI proposes a possible solution to this challenge (Schank, 1982). In a CBR framework, a new problem is solved by retrieving other *similar* problems and reusing their solution to derive a solution for the given problem. In other words, models, instead of memorizing patterns in its parameters, can instead reason with the reasoning patterns of other similar queries, retrieved dynamically during inference.

Inspired by CBR, this paper introduces a semiparametric model (CBR-SUBG) for question answering (QA) over KBs with a nonparametric component, that for each query, dynamically retrieves other similar k-nearest neighbor (KNN) queries from the training set. To retrieve similar queries, we use masked sentence representation of the query Soares et al. (2019) obtained from pre-trained language models.

We hypothesize that the reasoning patterns required for answering similar queries reoccur within the subgraph neighborhood of entities present in those queries (Figure 5.1). The answer nodes for each query are also analogously nestled within the reasoning patterns (marked as ☆ in Figure 5.1) of the query subgraphs, i.e. they have similar neighborhoods. However, we do not have annotated reasoning patterns that could be used to search for the answer node. Moreover, a subgraph can have tens of thousands of entity nodes. How do we still identify the answer nodes in the query subgraph?

To identify the answer nodes, our model has a parametric component comprising of a graph neural network (GNN) that is trained to identify the (latent) reasoning patterns from the subgraphs of KNN queries and apply it to the subgraph of the target query. GNNs have been shown to be effective in encoding structural properties of local neighborhoods in the node representations (Duvenaud et al., 2015; Kipf and Welling, 2017). We leverage node representations obtained from GNNs for finding answer nodes. Specifically, the answer nodes are identified by performing a nearest neighbor search for finding the most similar nodes in the query subgraph w.r.t the representation of answer nodes in the KNN subgraph. The parametric model is trained via contrastive learning (§5.3.3) Chopra et al. (2005); Gutmann and Hyvärinen (2010).

A practical challenge for KBQA models is to select a compact subgraph for a query. The goal is to ensure that the subgraph has high recall and is small enough to fit into GPU memory for gradient-based learning. Many KBQA methods usually consider few hops of edges around entities as the query subgraph Neelakantan et al. (2015b); Saxena et al. (2020) leading to query-independent and (often) large subgraphs, because of

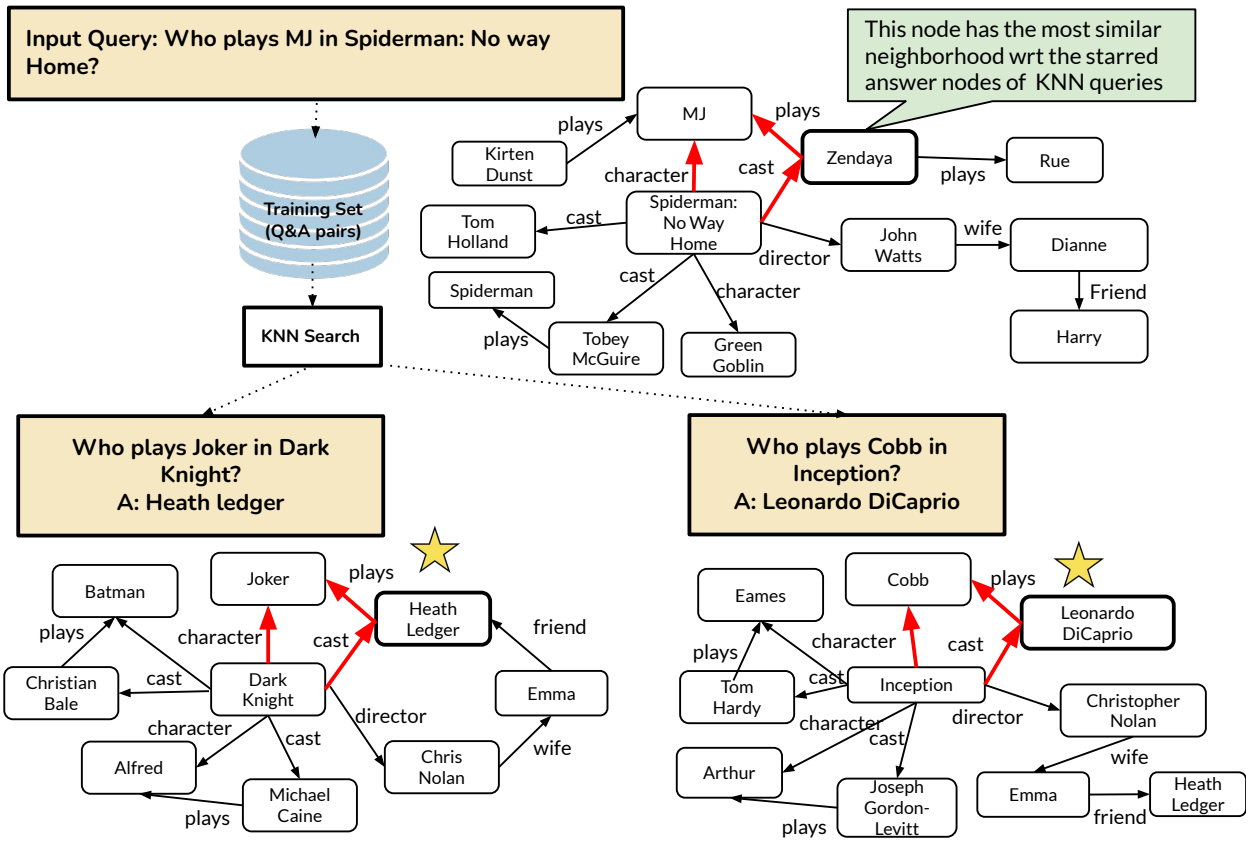


Figure 5.1: Figure shows an input query and two queries in the training set that are similar to the input query. The relevant subgraph for each query (query subgraph) is also shown. Note that the "reasoning patterns" required to answer the queries (edges in red) repeats in the subgraphs of each query. Also note, the corresponding answer nodes (marked as 'star') are analogously located within the reasoning patterns in each subgraph. Thus the answer node can be found by identifying the node in the query subgraph that is most similar to the answer nodes in the subgraph of KNN queries.

the presence of hub nodes in large KBs. We propose an *adaptive* subgraph collection method tailored for each query where we use our nonparametric approach of retrieving KNN queries to help gather the query subgraph leading to compact subgraphs with higher recall of reasoning patterns (§ 5.3.2).

An important property of nonparametric models is its ablility to grow and reason with new data. Being true to its nonparametric design, CBR-SUBG uses sparse representations of entities that makes it easy to represent new entities. Moreover, we also demonstrate that the performance of CBR-SUBG improves as more evidence is retrieved, suggesting that CBR-SUBG can reason with new evidence.

Contributions. To summarize, this paper introduces CBR-SUBG, a semiparametric model for weakly-supervised KBQA that retrieves similar queries and utilizes the similarities in graph structure of local subgraphs to answer a query (§**??**). We also propose a practical algorithm for gathering query-specific subgraphs that utilizes the retrieved KNN queries to produce compact query-specific subgraphs (§5.3.2). We show that CBR-SUBG can model (latent) subgraph reasoning patterns (§5.4.1), more effectively than parametric models; can reason with new entities (§5.4.1) and new evidence (§5.4.3). Lastly, we perform competitively with state-of-the-art KBQA models on multiple benchmarks. For example, on the FreebaseQA dataset Jiang et al. (2019), we outperform most competitive baseline by 14.45 points.

5.2 Related Work

Recently, Das et al. (2020a) proposed a CBR-based technique for KG completion. However, their work has several limitations. Firstly, it can only model simple linear chains. Secondly, it uses exact symbolic matching to find similarities in patterns between cases and the query. Lastly, it cannot handle natural language queries and only works with structured slot-filling queries. In contrast, CBR-SUBG can model arbitrary reasoning patterns; uses soft-matching by comparing representations of answer nodes and can handle natural language queries. Lastly, our method outperforms Das et al. (2020a) on various benchmarks. A follow up work of Das et al. (2021) proposed a CBR model that can handle natural language queries, however that work requires the availability of annotated reasoning patterns for training, an important distinction from our work that does not need any annotation of reasoning patterns.

Two classic semiparametric models for KBQA are GraftNet Sun et al. (2018) and PullNet Sun et al. (2019a) where they like us, provide both a mechanism of collecting a query-subgraph and reasoning over them. For their nonparametric component, these work employ a retrieval process where a parametric model classifies which edges would be relevant to the query. Being parametric, these models cannot generalize to new type of questions without re-training the model parameters. However, our nonparametric approach will work as it retrieves similar queries on-the-fly. For their reasoning model, both works use a graph convolution model and treat the answer prediction as a node classification task. However, unlike us they do not reason with subgraphs of similar KNN queries. Lastly, we compare extensively with them and outperform them on multiple benchmarks. Our model is also related to Teru et al. (2020a) as it explores KB reasoning in an inductive setting. They also have a sparse representation of entities. However, the task they consider is predicting a KB relation between two nodes which is an easier task than performing KBQA using natural language queries. Our approach also has similarities with retriever-reader architecture for open-domain QA over text Chen et al. (2017) where a retriever selects evidence specific to the query and the reader reasons with them to produce the answer. A model like CBR-SUBG is possible for KBQA because of tremendous progress made in graph representation learning (Kipf and Welling, 2017; Velickovic et al., 2018; Schlichtkrull et al., 2018, inter-alia).

CBR-SUBG shares similarities with the RETRIEVE-AND-EDIT framework Hashimoto et al. (2018) which utilizes retrieved nearest neighbor for structured prediction. However, unlike our method they only retrieve a single nearest neighbor and will unlikely be able to generate programs for questions requiring relations from multiple nearest neighbors. There has also been a lot of recent work in general NLP which uses KNN-based approaches. For example, Khandelwal et al. (2020) demonstrate improvements in language modeling by utilizing explicit examples from training data. There has been work in machine translation Gu et al. (2018); Khandelwal et al. (2021) that uses nearest neighbor translation pair to guide the decoding process.

5.3 Model

This section describes the nonparametric and parametric components of CBR-SUBG. In CBR, a case is defined as an abstract representation of a problem along with its solution. In our KBQA setting, a case is a natural language query along with its answer. Note in KBQA, answers are entities in the KB (or nodes in KG).

Task Description. Let q be a natural language query and let $\mathcal{K}$ be a symbolic KG that needs to be queried to retrieve an answer list $\mathcal{A}$ containing the answer(s) for q. We assume access to a training set $\mathcal{D} = \{(q_1, \mathcal{A}_1), (q_2, \mathcal{A}_2), \ldots (q_N, \mathcal{A}_N)\}$ of query-answer pairs where $\mathcal{A}_i$ denote the list of answer nodes for q_i. The training set $\mathcal{D}$ forms the 'case-base'. The reasoning pattern (a.k.a. logical form) required to answer a query are the set of KG edges required to deduce the answer to q. Let P_q denote this set of edges. For example, in Figure 5.1, for $q =$ "Who plays 'MJ' in No Way Home?", P_q = {(MJ, played_by, Zendaya), (No Way Home, has_character, MJ), (No Way Home, cast, Zendaya)}. We define reasoning pattern 'type' for a pattern as the set of edges where the entities have been replaced by free variables. For example, $T(P_q)$ = {(M, played_by, Z), (S, has_character, M), (S, cast, Z)}. It should be noted that CBR-SUBG does not assume access to annotated P_q.

For input q and $\mathcal{K}$, CBR-SUBG first retrieves k similar query-answer(s) w.r.t q from $\mathcal{D}$ (§ 5.3.1). Denote this

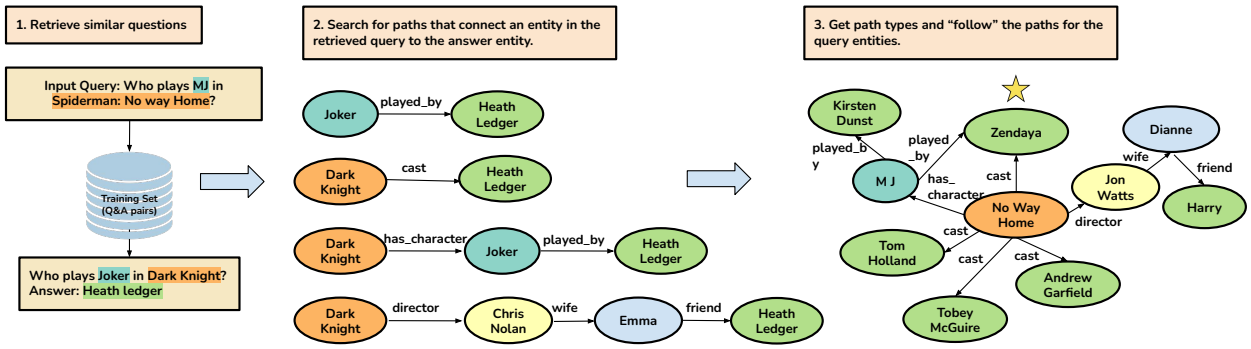


Figure 5.2: Figure shows the query-subgraph selection procedure with 1-nearest neighbor retrieved query. Graph paths connecting the entities in the retrieved query and its answer are collected. Next the sequence of relations (path types) are gathered and are then followed starting from the entity in the given query. All the edges spanned by this process are collected to form the query-specific subgraph. This process is repeated for each of the k retrieved queries.

retrieved set as $\text{kNN}_q \subset \mathcal{D}$. Next, CBR-SUBG finds query-specific subgraphs $\mathcal{K}_{q_i}$ for each query in $\{q\} \cup \text{kNN}_q$ (§ 5.3.2). According to the CBR hypothesis, the reasoning required to solve a new problem will be similar to the reasoning required to solve other similar problems. Similarly for our KBQA setting, we hypothesize that the reasoning pattern type, $T(P_q)$ repeats across the neighborhood of query subgraphs of kNN_q. Next, CBR-SUBG uses graph neural networks (GNNs) to encode local structure into node representations (§ 5.3.3). Now, if the CBR hypothesis holds true, then the representation of the answer nodes in each query subgraphs will be similar as the local structure around them share similarities. Hence, the answer node of the given query q can be identified by searching for the node that has the most similar representations to the answer nodes in the query subgraphs of kNN_q.

5.3.1 Retrieval of Similar Cases

Given the input query q, CBR-SUBG first retrieves other similar cases from the training set. We represent the query by embeddings obtained from large pre-trained language models Liu et al. (2019). Inspired by the recent advances in neural dense passage retrieval Karpukhin et al. (2020), we use a ROBERTA-base encoder to encode each question independently. A single representation is obtained by mean pooling over the token-level representations.

Generally, we want to retrieve questions that express similar relations rather than retrieving questions that are about similar entities. For example, for the query, 'Who played Natalie Portman in Star Wars?', we would like to retrieve queries such as 'Who played Ken Barlow in Coronation St.?' instead of 'What sports does Natalie Portman like to play?'. To obtain entity-agnostic representation, we replace the entity spans with a special '[MASK]' token in the query, i.e. the original query becomes 'Who played [MASK] in [MASK]?'. The entity masking strategy has previously been successfully used in learning entity-independent relational representations Soares et al. (2019). The similarity score between two queries is given by the inner product between their normalized vector representations (cosine similarity). We pre-compute the representations of queries in the train set. During inference, the most similar query representations are obtained by doing a nearest neighbor search over the representations stored in the case-base.

5.3.2 Query-subgraph Selection

A practical challenge for KBQA models is to select a subgraph around the entities present in the query. The goal is to ensure that the necessary reasoning patterns and answers are included while producing a graph small enough to fit into GPU memory for gradient-based learning. A naïve strategy to select a subgraph is to consider all edges in 2-3 hops around the query entities. This strategy leads to subgraphs which are independent of the query. Moreover, in large KGs like Freebase Bollacker et al. (2008), considering the

full 2 or 3-hop subgraph often leads to accumulation of millions of edges because of the presence of hub nodes.

We propose a nonparametric approach of query subgraph collection that utilizes the retrieved cases, kNN_q, from the last step (Figure 5.2). For each of the retrieved case, chains of edges (or paths) in the graph that connect the entity in the retrieved query to its answers are collected by doing a depth-first search. (Note, since the retrieved queries are from the training set, we know the answer to them). Next, the sequence of relations are collected from each chain and they are followed starting from the entities of the input query. If a chain of relations do not exist from the query, then they are simply ignored. This process is repeated for each of the retrieved cases. Note that, not all chains collected from the nearest neighbors are meaningful for the query. For example, the last (3-hop) chain collected in Figure 5.2 is not relevant for answering the query and even though it ended at the answer for the retrieved query, the same is not the case for the input query. Such paths are often referred to as spurious paths or evidence He et al. (2021). All the edges gathered by following this process form the subgraph for the input query. The underlying idea behind the subgraph selection procedure is simple — since the paths connect queries and answers of similar queries, they should also be relevant for answering the given query.

There is a class of prior work such as Graft-Net Sun et al. (2018) and PullNet Sun et al. (2019a) that learn a parametric model to choose a query-specific subgraph. These models employ a retrieval process where a parametric model classifies which edges would be relevant to the query. Being parametric, these models cannot generalize to new type of questions without re-training the model parameters. However, our nonparametric approach will work as long as it has access to similar queries, which it can retrieve on-the-fly. Our subgraph selection procedure is similar to the approach proposed in Das et al. (2020a). However, Das et al. (2020a) do not use this approach to collect a query-specific subgraph. Rather it uses each of the paths to *independently* predict the answer to a query. In contrast, we collect all edges in the path to form a subgraph and then reason jointly over the subgraph of the query as well as the subgraph of retrieved cases as detailed in the next sub-section.

5.3.3 Reasoning over Multiple Subgraphs

This section describes how CBR-SUBG reasons across the subgraphs of the given query and the subgraphs of the retrieved cases. We use graph neural networks (GNNs) Scarselli et al. (2008) to encode the local structure into the node representations of each subgraph. During training, the answer node representations of different subgraphs are made more similar to each other in comparison to other non-answer nodes. Inference reduces to searching for the most similar node in the query subgraph w.r.t the answer nodes in the KNN-subgraphs.

Modern GNNs employ a neighborhood aggregation strategy (message passing) where a node representation is iteratively updated by aggregating representations from its neighbors **?**. Let $\mathcal{G}_q = (V_q, E_q)$ represent the subgraph for a query q obtained from (§5.3.2). Let $\boldsymbol{X}_v$ denote the node feature vectors for each $v \in V_q$.

Input node representations. A property of nonparametric models is its ability to represent, reason and grow with new data. Knowledge graphs store facts about the world and as the world evolves, new entities and facts are added to the KG. Models developed for KG reasoning (Bordes et al., 2013; Schlichtkrull et al., 2018; Sun et al., 2019b, inter-alia) learn dense representations of a fixed vocabulary of entities and are hence unable to handle new entities added to the KG. Following our nonparametric design principles, each entity node is represented as a sparse vector of its outogoing edge (relation) types, i.e. $\boldsymbol{x}_v \in \{0,1\}^{|\mathcal{R}|}$ where $\mathcal{R}$ denotes the set of relation types in the KG. If entity x_v has m distinct outgoing edge types, then the dimension corresponding to those types are set to 1. This is an extremely simple and flexible way of representing entities which also expresses the local structural information around each node. Also note that, as new entities are added or new facts are updated about an entity, the sparse representation makes it very easy to represent new entities or update existing embeddings.

Relative distance embedding. Each query-specific subgraph $\mathcal{G}_q$ has a few special entities — the entities present in the input query. This is because the reasoning pattern is usually in the immediate subgraph surrounding the query entity. We treat the query entities as 'center' entities and append a relative distance

embedding to every other node in $\mathcal{G}_q$ Zhang and Chen (2018); Teru et al. (2020a). Specifically, for each node, the representation $\boldsymbol{x}_v$ is appended with an one-hot distance embedding $\boldsymbol{x_d} \in \{0,1\}^{|d|}$ where the component corresponding to the shortest distance from the query entity is set to 1. In practice, we consider subgraphs upto 3-hops from the query entities, i.e. $d = 4$. For queries with multiple query entities, the minimum distance is considered.

Message passing. Our GNN uses the graph structure and the sparse input node features $\boldsymbol{X}_v$ to learn intermediate node features capturing the local structure within them. We follow the general message-passing scheme where a node representation is iteratively updated by combining it with aggregation of it's neighbors' representation Xu et al. (2019). In particular, the l^{th} layer of a GNN is,

$$\boldsymbol{a}_v^l = \text{AGGREGATE}^l\left(\left\{\boldsymbol{h}_s^{l-1} : s \in \mathcal{N}(v)\right\}, \boldsymbol{h}_v^{l-1}\right), \tag{5.1}$$

$$\boldsymbol{h}_v^l = \text{COMBINE}^l\left(\boldsymbol{h}_v^{l-1}, \boldsymbol{a}_v^l\right), \tag{5.2}$$

where, $\boldsymbol{a}_v^l$ denote the aggregated message from the neighbors of node v, $\boldsymbol{h}_v^l$ denote the node representation of node v in the l-th layer and $\mathcal{N}(v)$ denotes the neighboring nodes of v. Since KGs are heterogenous graphs with labelled edges, we adopt the widely used multi-relational R-GCN model model Schlichtkrull et al. (2018) which defines the aggregate step as: $\boldsymbol{a}_v^l = \sum_{r=1}^{\mathcal{R}} \sum_{s \in \mathcal{N}_r(v)} \frac{1}{|\mathcal{N}_r(v)|} W_r^l h_s^{l-1}$ and the combine step as $h_v^l = \text{ReLU}(W_{\text{self}}^l h_v^{l-1} + \boldsymbol{a}_v^l)$. For each answer node, we consider the representation obtained from the last layer.

Training. Let a_i, a_j be an answer node in the corresponding query-subgraphs of q_i and q_j (i.e. $\mathcal{G}_{q_i}$, $\mathcal{G}_{q_j}$) respectively. Let $\text{sim}(\boldsymbol{a}_i, \boldsymbol{a}_j) = \boldsymbol{a}_i^\top \boldsymbol{a}_j / \|\boldsymbol{a}_i\| \|\boldsymbol{a}_j\|$ denote the inner product between ℓ_2 normalized answer representations (i.e. cosine similarity). In general there can be multiple answer nodes for a query. Let $\mathcal{A}_j$ denote the set of all answer nodes for query q_j in its subgraph $\mathcal{G}_{q_j}$. Let $\text{sim}(\boldsymbol{a}_i, \mathcal{A}_j) = \frac{1}{|\mathcal{A}_j|} \sum_{a_j \in \mathcal{A}_j} \text{sim}(\boldsymbol{a}_i, \boldsymbol{a}_j)$, i.e. $\text{sim}(\boldsymbol{a}_i, \mathcal{A}_j)$ represents the mean of the scores between a_{q_i} and all answer nodes in $\mathcal{G}_{q_j}$. We aggregate the similarity score from all retrieved queries kNN_{q_i} for the current query q_i.

The loss function we use is,

$$-\log \frac{\sum_{a_i \in \mathcal{A}_i} \exp(\sum_{q_j \in \text{KNN}_{q_i}} \text{sim}(\boldsymbol{a}_i, \mathcal{A}_j)/\tau)}{\sum_{x_i \in \mathcal{V}(\mathcal{G}_{q_i})} \exp(\sum_{q_j \in \text{KNN}_{q_i}} \text{sim}(\boldsymbol{x}_i, \mathcal{A}_j)/\tau)}, \tag{5.3}$$

where, $\mathcal{A}_j$ denotes the set of all answer nodes in $\mathcal{G}_{q_j}$ for a $q_j \in \text{kNN}_{q_i}$, x_i goes over all nodes in query-subgraph $\mathcal{G}_{q_i}$ and τ denotes a temperature parameter. In other words, the loss encourages the answer nodes in $\mathcal{G}_{q_i}$ to be scored higher than all other nodes in $\mathcal{G}_{q_i}$ w.r.t the answer nodes in the retrieved query subgraphs. This loss is an extension of the the normalized temperature-scaled cross entropy loss (*NT-Xent*) used in Chen et al. (2020).

Inference. During inference, message passing is run over each of the query-subgraph $\mathcal{G}_{q_i}$ and the subgraphs $\mathcal{G}_{q_j}$ of its k retrieved queries $q_j \in \text{kNN}_{q_i}$ to obtain the node representations and the highest scoring node in $\mathcal{G}_{q_i}$ w.r.t all the answer nodes in the retrieved query sub-graphs is returned as the answer.

$$\boldsymbol{a}_i = \underset{x_i}{\text{argmax}} \sum_{x_i \in \mathcal{V}(\mathcal{G}_{q_i})} \exp\left(\sum_{q_j \in \text{kNN}_{q_i}} \text{sim}(\boldsymbol{x}_i, \mathcal{A}_j)\right) \tag{5.4}$$

5.4 Experiments

In this section, we demonstrate the effectiveness of the semiparametric approach of CBR-SUBG and show that the nonparametric and parametric component offer complementary strengths. For example, we show that the model performance improves as more evidence is dynamically retrieved by the nonparametric component (§5.4.3). Similarly, CBR-SUBG can handle queries requiring reasoning patterns more complex than simple chains (i.e. subgraphs) because of the inductive bias provided by GNNs (§5.4.1). It can handle new and unseen entities because of the sparse entity input features as a part of its design (§5.4.1). We also show that the nonparametric subgraph selection of CBR-SUBG allows us to operate over a massive real-world

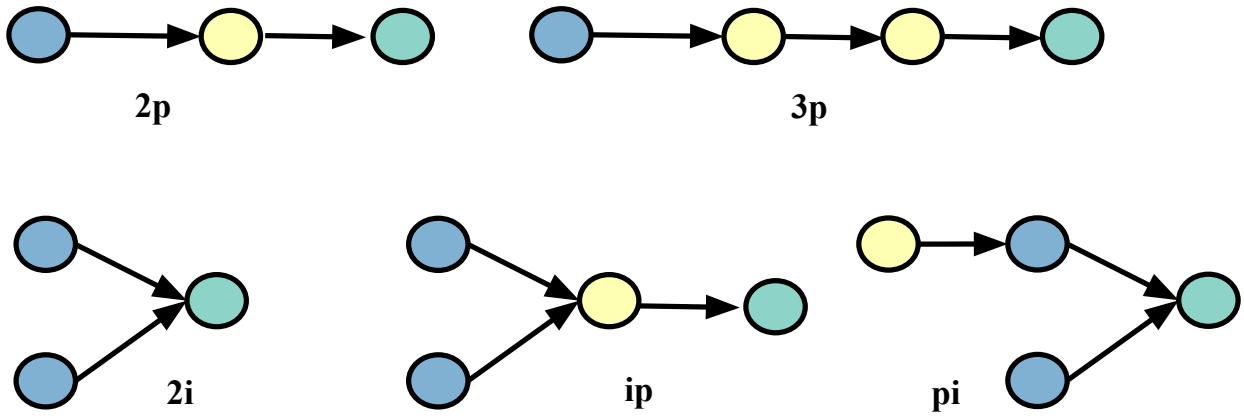


Figure 5.3: Various shapes of reasoning pattern types considered.

KG (full Freebase KG) and obtain very competitive performance on several benchmark datasets including WebQuestionsSP Yih et al. (2016), FreebaseQA Jiang et al. (2019) and MetaQA Zhang et al. (2018b).

5.4.1 Reasoning over Complex Patterns

We want to test whether CBR-SUBG can answer queries requiring complex reasoning patterns. Note that, the reasoning patterns are always latent to the model, i.e. the model has to answer a given query from the query-subgraph and the retrieved KNN-subgraphs *without* any knowledge of the structure of the pattern.

To test the model capacity to identify reasoning patterns, we devise a controlled setting in which the model has to infer reasoning patterns of various shapes (Figure 5.3), inspired by Ren et al. (2020). Note that in their work, the task was to execute the input structured query on an incomplete KB, i.e, the shape of the input patterns are known to the model. In contrast, in our setting, the model has to find the answer node (marked ●), which is nestled in each of the structured pattern without the knowledge of the pattern structure. Also note, there can be multiple nodes of the same type as the answer type, so the task cannot be completed by solving the easier task of determining entity types. Instead the model has to identify the specific ● nodes which are at the end of the reasoning patterns (there can be multiple ● nodes in the graph).

Data Generation Process. We first define a type system with a set of entity types $\mathcal{E}$ and relation types $\mathcal{R}$. The type-system also specifies a set of 'allowed relation types' between different pairs of entity types. For example, an 'employee' KB relation is defined between an 'organization' and 'people' entity types. Entities (or nodes $\mathcal{V}$) are then generated uniformly from the set of entity types $\mathcal{E}$. Next, relation edges (uniformly sampled from the allowed types) are joined between a pair of nodes with a probability p following the Erd$\mathcal{H}$os-Rényi model **?** of random graph generation. To ensure that models only rely on the graph structure, each graph has a 'unique' set of entities and no two graphs share entities. This also effectively tests how much the nonparametric property of CBR-SUBG can reason with unseen entities. More details regarding the hyperparameters $\mathcal{E}, \mathcal{R}, \mathcal{V}$ are included in the hyperparameter section.

Pattern Generation. A pattern is next sampled from the set of shapes shown in Figure 5.3. The sampled pattern merely suggests the structure of the desired pattern. '*Grounding*' a pattern shape involves assigning each nodes with an entity present in a generated graph. Similarly each edge of the pattern type is assigned a relation from the set of allowed relation types. After grounding the pattern, we "insert" the pattern in the graph. Since the nodes of the grounded pattern already exists in the graph, inserting a pattern in the graph amounts to adding the edges of the grounded pattern to the graph that already did not exist in it. We also define a 'pattern type' — that refers to a pattern whose edges have been assigned relation types but the nodes have not been assigned to specific entities (bottom-left corner in Figure 5.4). Each pattern type is assigned an identifier and queries with the same pattern type are grouped together.

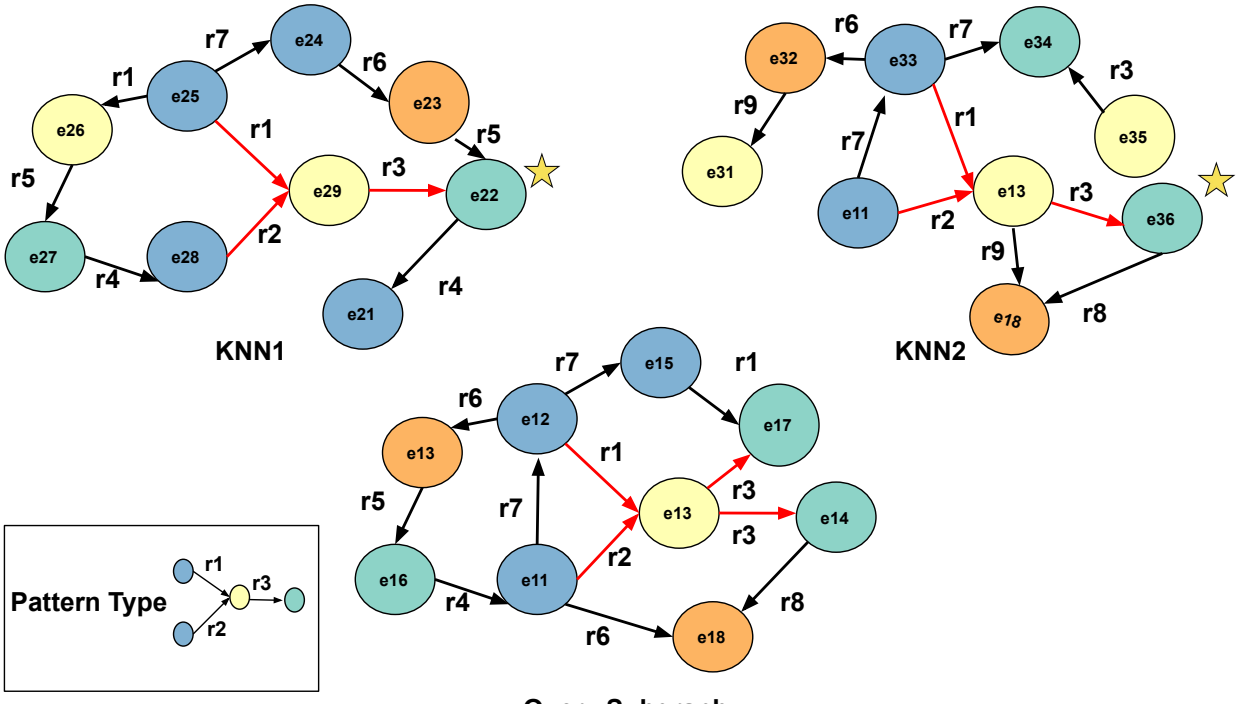


Figure 5.4: Setup for reasoning over subgraph patterns. Note that the same 'pattern type' repeats across subgraphs. Also note, that the query graph (bottom) has two answer nodes (e14, e17). Since every subgraph has its own set of unique entities, hence a model has to reason over the similarities in graph structures to find the answer node in the query-subgraph.

Model	2p	3p	2i	ip	pi	avg.
CBR-SUBG (No training)	68.56	84.35	23.00	34.85	35.35	47.28
GNN + TransE	80.03	74.49	80.00	52.67	81.53	72.69
CBR Das et al. (2020a)	69.71	54.39	**100.00**	69.12	51.24	71.09
CBR-SUBG	**96.64**	**88.43**	90.46	**70.02**	**86.81**	**85.68**

Table 5.1: Strict Hits1(%) for predicting *all* the answer nodes correctly. CBR-SUBG without any training performs decently suggesting that it has the right inductive bias for the task. On training, the performance improves on all subgraph patterns.

We generate 1000 graphs in each of the training, validation and test sets. We generate 200 pattern types whose shapes are uniformly sampled from the 5 shapes shown in Figure 5.3. Therefore, there are around 40 examples of each pattern shape and around 5 examples of each pattern type. This is consistent with real-world setting where a model will encounter a reasoning pattern only very few times during training. For a query with a particular pattern type, other training queries with the same pattern type form its nearest neighbors.

Hyperparameters For MetaQA, we use 3 GCN layers with GCN layer dimension of 32. For training we have used 5 nearest neighbors and 10 are used for evaluation for the 1-hop, 2-hop and 3-hop queries.We optimize the loss using Adam Optimizer with beta1 of 0.9, beta2 of 0.999 and epsilon of 1e-8. As well as the learning rate is set to be 0.00099 with temperature value of 0.0382 (1-hop), 0.0628 (2-hop) ,0.0779 (3-hop). All the models are trained for 5 epochs.

Similarly for WebQSP, we use 3 GCN layers with GCN layer dimension of 32. But for training we used 10 nearest neighbors and 5 are used for evaluation. We optimize the loss using Adam Optimizer with beta1 of 0.9, beta2 of 0.999 and epsilon of 1e-8. As well as a learning rate of 0.0024 and temperature of 0.0645 is used. Though the model is trained for about 30 epochs.

Dataset	Train	Dev	Test
MetaQA 1-hop	96,106	9,992	9,947
MetaQA 2-hop	118,980	14,872	14,872
MetaQA 3-hop	114,196	14,274	14,274
WebQSP	2,848	250	1,639
FreebaseQA	20,358	2308	3996

Table 5.2: Dataset Statistics

Model	**MetaQA**			**WebQSP**
	1-hop	**2-hop**	**3-hop**	
KVMemNN Miller et al. (2016b)	95.8	25.1	10.1	46.7
GraftNet Sun et al. (2018)	97.0	94.8	77.7	66.4
PullNet Sun et al. (2019a)	97.0	99.9	91.4	68.1
SRN Qiu et al. (2020)	97.0	95.1	75.2	-
ReifKB Cohen et al. (2020)	96.2	81.1	72.3	52.7
EmbedKGQA Saxena et al. (2020)	**97.5**	98.8	94.8	66.6
NSM He et al. (2021)	97.2	99.9	98.9	**74.3**
CBR-SUBG (Ours)	97.1	99.8	**99.3**	71.9

Table 5.3: Performance on WEBQUESTIONSSP and MetaQA benchmarks.

Baselines. Because of the inductive nature of this task where only new entities are seen at test time, most parametric KG reasoning algorithms Bordes et al. (2013); Yang et al. (2015); Sun et al. (2019b) will not work out of the box. We extend the widely used KG reasoning model — TransE Bordes et al. (2013) to work in the inductive setting. Specifically, instead of having a fixed vocabulary of entities, the objective function is computed on the dense representation obtained from the output of the GNN layers. This also makes the comparision with CBR-SUBG fair since it also operates on the same representations albeit with a contrastive loss. KG completion algorithms also need a query-relation as input. Each pattern type for a query serves as the query relation. Apart from the parametric baseline, we also test the nonparametric approached proposed by Das et al. (2020a) (CBR in Table 5.1). Comparing CBR-SUBG to CBR will help us understand the importance of modeling subgraph patterns rather than simple chains.

Does CBR-SUBG have the right inductive bias? The first research question that we try to answer is, if CBR-SUBG has the right inductive bias for this task. We test CBR-SUBG that has undergone *no training*, i.e. the parameters of the GNN are randomly initialized. Note the model still takes as input the sparse representation of entities. This experiment will help us answer if the node representations actually capture the local structure around them and whether the answer node can be found by doing a search w.r.t the answer nodes in the KNN-query subgraphs.

Table 5.1 reports the strict hits@1 on this task, i.e. to score a query correctly, a model has to identify and rank *all* answer nodes above all other nodes in the graph. The first row of Table 5.1 shows the results. For comparison, a random performance on this task is $\frac{1}{|\mathcal{V}|} = \frac{1}{120} = 0.83\%$. As it is clear from the results, an un-trained CBR-SUBG achieves performance much higher than random performance. Its quite high for the simple 2p and 3p patterns. For other patterns that need the more complicated intersection operation, the performance degrades, but is still much higher than random.

Our Results. On training CBR-SUBG, the performance of the model drastically improves for each pattern type reaching an average performance of 85.68%. The performance on pattern types which are more complex than chains (ip, pi) etc are worse than chain-type patterns (2p, 3p) suggesting that our task is non-trivial[1].
On comparison to parametric model. This experiment helps us understand whether a model can learn to memorize and store patterns effectively (for each query relations) when it has seen few examples of that

[1]We will release the code, dataset and data generation pipeline for reproducibility and further research

Model	Accuracy
KB-only models	
HR-BiLSTM Yu et al. (2017)	28.40
KBQA-Adapter Wu et al. (2019)	28.78
KEQA Huang et al. (2019)	28.73
FOFE Jiang et al. (2019)	37.00
BuboQA Mohammed et al. (2018)	38.25
CBR Das et al. (2020a)	25.40
CBR-SUBG (Ours)	**52.07**
LM pre-training + KB	
EAE Févry et al. (2020)	53.4
FAE Verga et al. (2020)	**63.3**

Table 5.4: Top-1 % accuracy on the FREEBASEQA dataset. The top section reports performance of models that operate only on KBs. The bottom section reports performance on models that also use additional knowledge stored in large language models.

pattern during training. Row 2 of Table 5.1 shows the performance of GNN + TransE model. We find that the parametric model performs worse than CBR-SUBG on all the query types reaching an average performance of 13% point below CBR-SUBG. This shows that a semiparametric model with a nonparametric component that retrieves similar queries at inference can make it easier for the model to reason effectively. In practice, we had to train this model for a much longer time than training CBR-SUBG.

On comparison to path-based model. From Table 5.1, we can see that CBR-SUBG outperforms CBR Das et al. (2020a) by more than 14% points suggesting that reasoning over subgraphs is a more powerful approach that reasoning with each paths independently. On the '2i' pattern, CBR outperforms CBR-SUBG since '2i' can be seen as 2 independent paths intersecting at one node and CBR is able to model that perfectly. However, when the pattern needs composition and intersection and path-traversal, CBR struggles and performs much worse.

5.4.2 Performance on benchmark datasets

Next, we test the performance of CBR-SUBG on various KBQA benchmarks — MetaQA Zhang et al. (2018b), WEBQUESTIONSSP Yih et al. (2016) and FREEBASEQA Jiang et al. (2019). MetaQA comes with its own KB. For other datasets, the underlying KB is the full Freebase KB containing over 45 million entities (nodes) and 3 billion facts (edges). Table 5.2 summarizes the basic statistics of the datasets used in our experiments.

Our main baselines are the two semiparametric models that provide both a mechanism to gather query subgraphs for a given query and reason over them to find the answer — GraftNet Sun et al. (2018), PullNet Sun et al. (2019a). GraftNet uses personalized page rank to determine which edges are relevant for a particular query and PullNet uses a multi-step retriever that at each step, classifies if an edge is relevant to the current representation of the query. For their reasoning model, both works use a graph convolution model and treat the answer prediction as a node classification task. However unlike us, they do not use query-subgraphs of KNN queries. Followup KBQA works (Saxena et al., 2020; He et al., 2021, inter-alia) use the query-specific graphs provided by GraftNet from their open-source code and do not provide a mechanism to gather query-specific subgraphs. However, for completeness, we report and compare with those methods as well.

Table 5.3 reports the performance on WEBQUESTIONSSP and all three partitions of MetaQA. When compared to GraftNet and PullNet, CBR-SUBG performs much better on an average on both the datasets. On the more challenging 3-hop subset of MetaQA, CBR-SUBG outperforms PullNet by more than 7 points and GraftNet by more than 15 points. This shows that even though these two models use a GNN for reasoning, using information from subgraphs from similar KNN queries leads to much better performance. On WEBQUESTIONSSP, we outperform all models except the recently proposed NSM model He et al. (2021).

Subgraph	WebQSP	MetaQA-3
GraftNet	65.61%	96.90%
Adaptive	71.92%	99.30%

Table 5.5: Performance of CBR-SUBG with adaptive subgraph and GraftNet subgraph

Subgraph	#edges	#relations	#entities	coverage(%)
WEBQUESTIONSSP				
Graft-net	4306.00	294.69	1447.68	89.93%
CBR-SUBG	2234.35	40.38	1627.37	94.30%
% diff	-48.11%	-86.5%	+12.4%	+4.85%
MetaQA-3				
Graft-net	1153.0	18.00	497.00	99%
CBR-SUBG	89.21	4.72	77.52	99.9%
% diff	-92.21%	-73.78%	-84.40%	+0.91%

Table 5.6: Our adaptive subgraph collection strategy produces a compact subgraph for a query while increasing recall.

But as we noted before, NSM operates on the subgraph created by GraftNet and does not provide any particular mechanism to create its own query-specific subgraph (an important contribution of our model). Moreover NSM is a parametric model and will not have some advantages of nonparametric architectures such as ability to handle new entities and reasoning with more data. Table 5.4 reports the results on the FREEBASEQA dataset, which contains real trivia questions obtained from various trivia competitions. Thus the questions can be challenging in nature. We compare with other KBQA models reported in Han et al. (2020). Most of the models are pipelined KBQA systems that rely on relation extraction to map the query into a KB edge. CBR-SUBG outperforms all the models by a large margin. We also report the performance on two models that use large LMs and large-scale pre-training. CBR-SUBG, which only operates on the KB has a performance very close to the performance of Entity-as-Experts model Févry et al. (2020). We leave the integration of large LMs into our parametric reasoning component as future work.

5.4.3 Analysis

How effective is our adaptive subgraph collection strategy? Table 5.6 reports few average graph statistics for the query-subgraphs collected by our graph-collection strategy. We also compare to GraftNet's subgraphs[2]. As can be seen, our adaptive graph collection strategy produces much more compact and smaller graphs *while* increasing recall of answers. We also consistently find that our graph contains relations which is more relevant to the questions than the subgraph produced by GraftNet[3] Table 5.5 reports the performance of CBR-SUBG when trained and tested on the subgraph obtained from Graftnet and our adaptive procedure, demonstrating the effectiveness of our adaptive subgraph collection method.

Can CBR-SUBG reason with more evidence? A desirable property of nonparametric models is to be able to 'improve' its prediction as more evidence is made available. We test CBR-SUBG by taking a trained model and issuing it an increasing number of nearest neighbor queries. As we see from Figure 5.5, the performance of CBR-SUBG drastically improves as we increase the number of nearest neighbors from 1 to 7 and then increases at a lower rate and converges at around 10 nearest neighbors. This is because, the model has all the required information it needs from its nearest neighbors.

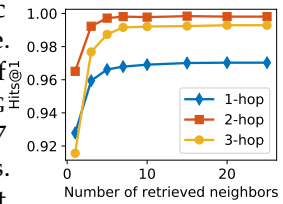


Figure 5.5: Performance of CBR-SUBG as more nearest neighbors are introduced at test-time. Results on different partitions of MetaQA

[2]Code and artifacts for PullNet is not available

[3]We will also released the collected subgraphs for all the datasets.

5.5 Conclusion

In this work, we explored a semiparametric approach for KBQA. We demonstrated CBR-SUBG poses several desirable properties approach in which nonparametric and parametric component offer complementary strengths. By retrieving similar queries and utilizing the similarities in graph structure of local subgraphs to answer a query, our approach is able to handle complex questions as well as generalize to new types of questions. Exploring different types of parametric models with different reasoning capabilities (LMs, GNNs, etc.) would be an interesting future research direction. Another avenue of potential research would be a never-ending learning type of system where we keeps adding newly discovered facts in the nonparametric part.

Bibliography

Agnar Aamodt and Enric Plaza. 1994. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI communications*.

Edoardo M Airoldi, David M Blei, Stephen E Fienberg, and Eric P Xing. 2008. Mixed membership stochastic blockmodels. *JMLR*.

Gabor Angeli and Christopher D Manning. 2014. Naturalli: Natural logic inference for common sense reasoning. In *EMNLP*.

Kai Bartlmae and Michael Riemenschneider. 2000. Case based reasoning for knowledge management in kdd projects. In *PAKM*.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on freebase from question-answer pairs. In *EMNLP*.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: A collaboratively created graph database for structuring human knowledge. In *ICDM*.

Antoine Bordes, Nicolas Usunier, Sumit Chopra, and Jason Weston. 2015. Large-scale simple question answering with memory networks. *arXiv preprint arXiv:1506.02075*.

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *Neurips*.

Antal van den Bosch, Bertjan Busser, Sander Canisius, and Walter Daelemans. 2007. An efficient memory-based morphosyntactic tagger and parser for dutch. *LOT Occasional Series*.

Guillaume Bouchard, Sameer Singh, and Theo Trouillon. 2015. On approximate reasoning capabilities of low-rank vector spaces. *AAAI Spring Symposium*.

Matthias Broecheler, Lilyana Mihalkova, and Lise Getoor. 2010. Probabilistic similarity logic. In *UAI*.

Ziqiang Cao, Wenjie Li, Sujian Li, and Furu Wei. 2018. Retrieve, rerank and rewrite: Soft template based neural summarization. In *ACL*.

Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R. Hruschka, Jr., and Tom M. Mitchell. 2010. Toward an Architecture for Never-ending Language Learning. In *AAAI*.

Kai-Wei Chang, Akshay Krishnamurthy, Alekh Agarwal, Hal Daume, and John Langford. 2015. Learning to search better than your teacher. In *ICML*.

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading wikipedia to answer open-domain questions. In *ACL*.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A simple framework for contrastive learning of visual representations. In *ICML*.

Wenhu Chen, Wenhan Xiong, Xifeng Yan, and William Wang. 2018. Variational knowledge graph reasoning. In *NAACL*.

Sumit Chopra, Raia Hadsell, and Yann LeCun. 2005. Learning a similarity metric discriminatively, with application to face verification. In *CVPR*.

William Cohen. 2016. Tensorlog: A differentiable deductive database. *arXiv:1605.06523*.

William W Cohen, Haitian Sun, R Alex Hofer, and Matthew Siegler. 2020. Scalable neural methods for reasoning with a symbolic knowledge base. *arXiv preprint arXiv:2002.06115*.

Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *ACL*.

Thomas Cover and Peter Hart. 1967. Nearest neighbor pattern classification. *IEEE transactions on information theory*.

Walter Daelemans, Jakub Zavrel, Peter Berck, and Steven Gillis. 1996. Mbt: A memory-based part of speech tagger-generator. In *WVLC*.

Rajarshi Das, Shehzaad Dhuliawala, Manzil Zaheer, and Andrew McCallum. 2019. Multi-step retriever-reader interaction for scalable open-domain question answering. In *ICLR*.

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

Rajarshi Das, Ameya Godbole, Shehzaad Dhuliawala, Manzil Zaheer, and Andrew McCallum. 2020a. A simple approach to case-based reasoning in knowledge bases. *arXiv preprint arXiv:2006.14198*.

Rajarshi Das, Ameya Godbole, Nicholas Monath, Manzil Zaheer, and Andrew McCallum. 2020b. Probabilistic case-based reasoning for open-world knowledge graph completion. *arXiv preprint arXiv:2010.03548*.

Rajarshi Das, Ameya Godbole, Ankita Naik, Elliot Tower, Manzil Zaheer, Robin Jia, Hannaneh Hajishirzi, and Andrew McCallum. 2022. Knowledge base question answering by case-based reasoning over subgraphs. *arXiv*.

Rajarshi Das, Arvind Neelakantan, David Belanger, and Andrew McCallum. 2017. Chains of reasoning over entities, relations, and text using recurrent neural networks. In *EACL*.

Rajarshi Das, Manzil Zaheer, Dung Thai, Ameya Godbole, Ethan Perez, Jay-Yoon Lee, Lizhen Tan, Lazaros Polymenakos, and Andrew McCallum. 2021. Case-based reasoning for natural language queries over knowledge bases. *arXiv preprint arXiv:2104.08762*.

Hal Daumé III, John Langford, and Daniel Marcu. 2009. Search-based structured prediction. *Machine learning*.

Hal Daumé III and Daniel Marcu. 2005. Learning as search optimization: Approximate large margin methods for structured prediction. In *ICML*.

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

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

W Dubitzky, AG Büchner, and FJ Azuaje. 1999. Viewing knowledge management as a case-based reasoning application. In *AAAI Workshop Technical Report*, pages 23–27.

David Duvenaud, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gómez-Bombarelli, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. 2015. Convolutional networks on graphs for learning molecular fingerprints. *arXiv preprint arXiv:1509.09292*.

Michael Evans and Timothy Swartz. 2000. *Approximating integrals via Monte Carlo and deterministic methods*. OUP Oxford.

Thibault Févry, Livio Baldini Soares, Nicholas FitzGerald, Eunsol Choi, and Tom Kwiatkowski. 2020. Entities as experts: Sparse memory access with entity supervision. In *EMNLP*.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*.

George Fishman. 2013. *Monte Carlo: concepts, algorithms, and applications*. Springer Science & Business Media.

Jonathan Frankle and Michael Carbin. 2019. The lottery ticket hypothesis: Finding sparse, trainable neural networks.

Daniel Furrer, Marc van Zee, Nathan Scales, and Nathanael Schärli. 2020. Compositional generalization in semantic parsing: Pre-training vs. specialized architectures. *arXiv preprint arXiv:2007.08970*.

Evgeniy Gabrilovich, Michael Ringgaard, and Amarnag Subramanya. 2013. Facc1: Freebase annotation of clueweb corpora, version 1 (release date 2013-06-26, format version 1, correction level 0).

Luis Galárraga, Christina Teflioudi, Katja Hose, and Fabian M Suchanek. 2015. Fast rule mining in ontological knowledge bases with amie+. In *VLDB*.

Luis Antonio Galárraga, Christina Teflioudi, Katja Hose, and Fabian Suchanek. 2013. Amie: association rule mining under incomplete evidence in ontological knowledge bases. In *WWW*.

Alberto Garcia-Duran and Mathias Niepert. 2018. Kblrn: End-to-end learning of knowledge base representations with latent, relational, and numerical features. In *UAI*.

Lise Getoor and Ben Taskar. 2007. *Introduction to statistical relational learning*. MIT press.

Alex Graves, Greg Wayne, and Ivo Danihelka. 2014. Neural turing machines. *arXiv:1410.5401*.

Spence Green, Nicholas Andrews, Matthew R Gormley, Mark Dredze, and Christopher D Manning. 2012. Entity clustering across languages. In *NAACL-HLT*.

Jiatao Gu, Yong Wang, Kyunghyun Cho, and Victor OK Li. 2018. Search engine guided neural machine translation. In *AAAI*.

Yu Gu, Sue E. Kase, Michelle Vanni, Brian M. Sadler, Percy Liang, Xifeng Yan, and Y. Su. 2021. Beyond i.i.d.: Three levels of generalization for question answering on knowledge bases. In *WWW*, volume abs/2011.07743.

Shu Guo, Quan Wang, Lihong Wang, Bin Wang, and Li Guo. 2016. Jointly embedding knowledge graphs and logical rules. In *EMNLP*.

Michael Gutmann and Aapo Hyvärinen. 2010. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *AIStats*.

Kelvin Guu, Tatsunori B. Hashimoto, Yonatan Oren, and Percy Liang. 2018. Generating sentences by editing prototypes. In *Transactions of the Association for Computational Linguistics (TACL)*.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. 2020. Realm: Retrieval-augmented language model pre-training. In *ICML*.

Kelvin Guu, John Miller, and Percy Liang. 2015. Traversing knowledge graphs in vector space. In *EMNLP*.

John Hammersley. 2013. *Monte carlo methods*. Springer Science & Business Media.

Namgi Han, Goran Topic, Hiroshi Noji, Hiroya Takamura, and Yusuke Miyao. 2020. An empirical analysis of existing systems and datasets toward general simple question answering. In *CoNLL*.

Tatsunori B Hashimoto, Kelvin Guu, Yonatan Oren, and Percy Liang. 2018. A retrieve-and-edit framework for predicting structured outputs. In *Neurips*.

Gaole He, Yunshi Lan, Jing Jiang, Wayne Xin Zhao, and Ji-Rong Wen. 2021. Improving multi-hop knowledge base question answering by learning intermediate supervision signals. In *WSDM*.

Geoffrey E Hinton and David C Plaut. 1987. Using fast weights to deblur old memories. In *Proceedings of the ninth annual conference of the Cognitive Science Society*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*.

Nabil Hossain, Marjan Ghazvininejad, and Luke Zettlemoyer. 2020. Simple and effective retrieve-edit-rerank text generation. In *ACL*.

Xiao Huang, Jingyuan Zhang, Dingcheng Li, and Ping Li. 2019. Knowledge graph embedding based question answering. In *WSDM*.

Wonseok Hwang, Jinyeong Yim, Seunghyun Park, and Minjoon Seo. 2019. A comprehensive exploration on wikisql with table-aware word contextualization. *arXiv preprint arXiv:1902.01069*.

Kelvin Jiang, Dekun Wu, and Hui Jiang. 2019. Freebaseqa: a new factoid qa data set matching trivia-style question-answer pairs with freebase. In *NAACL*.

Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *EMNLP*.

Charles Kemp, Joshua B Tenenbaum, Thomas L Griffiths, Takeshi Yamada, and Naonori Ueda. 2006. Learning systems of concepts with an infinite relational model. In *AAAI*.

Daniel Keysers, Nathanael Schärli, Nathan Scales, Hylke Buisman, Daniel Furrer, Sergii Kashubin, Nikola Momchev, Danila Sinopalnikov, Lukasz Stafiniak, Tibor Tihon, Dmitry Tsarkov, Xiao Wang, Marc van Zee, and Olivier Bousquet. 2020. Measuring compositional generalization: A comprehensive method on realistic data. In *ICLR*.

Urvashi Khandelwal, Angela Fan, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2021. Nearest neighbor machine translation. In *ICLR*.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2020. Generalization through memorization: Nearest neighbor language models. In *ICLR*.

Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. 2017. Overcoming catastrophic forgetting in neural networks. *PNAS*.

Ari Kobren, Nicholas Monath, Akshay Krishnamurthy, and Andrew McCallum. 2017. A hierarchical algorithm for extreme clustering. In *KDD*.

Stanley Kok and Pedro Domingos. 2007. Statistical predicate invention. In *ICML*.

Janet L Kolodner. 1983. Maintaining organization in a dynamic long-term memory. *Cognitive science*.

Brenden M Lake and Marco Baroni. 2018. Generalization without systematicity. In *ICML*.

Yunshi Lan and Jing Jiang. 2020. Query graph generation for answering multi-hop complex questions from knowledge bases. In *ACL*.

Yunshi Lan, Shuohang Wang, and Jing Jiang. 2019. Knowledge base question answering with topic units.

Juliana S Lancaster and Janet L Kolodner. 1987. Problem solving in a natural task as a function of experience. Technical report, Georgia Tech CS Department.

Ni Lao, Tom Mitchell, and William Cohen. 2011. Random walk inference and learning in a large scale knowledge base. In *EMNLP*.

David B Leake. 1996. Cbr in context: The present and future. *Case-based reasoning: Experiences, lessons, and future directions*.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature*.

Heeyoung Lee, Marta Recasens, Angel Chang, Mihai Surdeanu, and Dan Jurafsky. 2012. Joint entity and event coreference resolution across documents. In *EMNLP/CoNLL*.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020a. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *ACL*.

Patrick Lewis, Ethan Perez, Aleksandara Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020b. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Neurips*.

Patrick Lewis, Pontus Stenetorp, and Sebastian Riedel. 2020c. Question and answer test-train overlap in open-domain question answering datasets. *arXiv preprint arXiv:2008.02637*.

Patrick Lewis, Yuxiang Wu, Linqing Liu, Pasquale Minervini, Heinrich Küttler, Aleksandra Piktus, Pontus Stenetorp, and Sebastian Riedel. 2021. Paq: 65 million probably-asked questions and what you can do with them.

Belinda Z Li, Sewon Min, Srinivasan Iyer, Yashar Mehdad, and Wen-tau Yih. 2020. Efficient one-pass end-to-end entity linking for questions. In *EMNLP*.

Chen Liang, Jonathan Berant, Quoc Le, Kenneth D Forbus, and Ni Lao. 2016. Neural symbolic machines: Learning semantic parsers on freebase with weak supervision. *arXiv preprint arXiv:1611.00020*.

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

Xi Victoria Lin, Chenglong Wang, Luke Zettlemoyer, and Michael D Ernst. 2018b. Nl2bash: A corpus and semantic parser for natural language interface to the linux operating system. In *LREC*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Joao Loula, Marco Baroni, and Brenden M Lake. 2018. Rearranging the familiar: Testing compositional generalization in recurrent networks. In *EMNLP Blackbox NLP Workshop*.

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

Bill MacCartney and Christopher D Manning. 2007. Natural logic for textual inference. In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*. Association for Computational Linguistics.

John McCarthy. 1960. *Programs with common sense*. RLE and MIT Computation Center.

Michael McCloskey and Neal J Cohen. 1989. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*.

Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. 2016a. Key-value memory networks for directly reading documents. *EMNLP*.

Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. 2016b. Key-value memory networks for directly reading documents. In *EMNLP*.

Kurt Miller, Michael I Jordan, and Thomas L Griffiths. 2009. Nonparametric latent feature models for link prediction. In *Neurips*.

Bonan Min, Ralph Grishman, Li Wan, Chang Wang, and David Gondek. 2013. Distant supervision for relation extraction with an incomplete knowledge base. In *NAACL-HLT*.

Pasquale Minervini, Matko Bošnjak, Tim Rocktäschel, Sebastian Riedel, and Edward Grefenstette. 2020. Differentiable reasoning on large knowledge bases and natural language. In *AAAI*.

Pasquale Minervini, Thomas Demeester, Tim Rocktäschel, and Sebastian Riedel. 2017. Adversarial sets for regularising neural link predictors. *arXiv preprint arXiv:1707.07596*.

Mike Mintz, Steven Bills, Rion Snow, and Dan Jurafsky. 2009. Distant supervision for relation extraction without labeled data. In *ACL*.

Salman Mohammed, Peng Shi, and Jimmy Lin. 2018. Strong baselines for simple question answering over knowledge graphs with and without neural networks. In *NAACL*.

Nicholas Monath, Ari Kobren, Akshay Krishnamurthy, Michael R Glass, and Andrew McCallum. 2019. Scalable hierarchical clustering with tree grafting. In *KDD*.

Stephen Muggleton. 1995. Inverse entailment and progol. *New generation computing*.

Stephen Muggleton, Ramon Otero, and Alireza Tamaddoni-Nezhad. 1992. *Inductive logic programming*. Springer.

Stephen Mussmann and Stefano Ermon. 2016. Learning and inference via maximum inner product search. In *ICML*.

Arvind Neelakantan, Benjamin Roth, and Andrew McCallum. 2015a. Compositional vector space models for knowledge base completion. In *ACL*.

Arvind Neelakantan, Benjamin Roth, and Andrew McCallum. 2015b. Compositional vector space models for knowledge base completion. In *ACL*.

Ngonga Ngomo. 2018. 9th challenge on question answering over linked data (qald-9). *language*.

Maximilian Nickel, Xueyan Jiang, and Volker Tresp. 2014. Reducing the rank in relational factorization models by including observable patterns. In *NIPS*.

Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. 2011a. A three-way model for collective learning on multi-relational data. In *ICML*.

Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. 2011b. A three-way model for collective learning on multi-relational data. In *ICML*.

Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. 2012. Factorizing yago: scalable machine learning for linked data. In *WWW*.

Nils J Nilsson. 1991. Logic and artificial intelligence. *Artificial intelligence*.

Rodrigo Nogueira and Kyunghyun Cho. 2016. End-to-end goal-driven web navigation. In *NIPS*.

Gaurav Pandey, Danish Contractor, Vineet Kumar, and Sachindra Joshi. 2018. Exemplar encoder-decoder for neural conversation generation. In *ACL*.

Hao Peng, Ankur Parikh, Manaal Faruqui, Bhuwan Dhingra, and Dipanjan Das. 2019. Text generation with exemplar-based adaptive decoding. In *NAACL*.

Yunqi Qiu, Yuanzhuo Wang, Xiaolong Jin, and Kun Zhang. 2020. Stepwise reasoning for multi-relation question answering over knowledge graph with weak supervision. In *WSDM*.

J Ross Quinlan. 1990. Learning logical definitions from relations. *Machine learning*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *JMLR*.

Carl Edward Rasmussen. 2000. The infinite gaussian mixture model. In *Neurips*.

Hongyu Ren, Weihua Hu, and Jure Leskovec. 2020. Query2box: Reasoning over knowledge graphs in vector space using box embeddings. In *ICLR*.

Matthew Richardson and Pedro Domingos. 2006. Markov logic networks. *Machine learning*.

Sebastian Riedel, Limin Yao, Andrew McCallum, and Benjamin M. Marlin. 2013. Relation extraction with matrix factorization and universal schemas. In *NAACL*.

Edwina L Rissland. 1983. Examples in legal reasoning: Legal hypotheticals. In *IJCAI*.

Tim Rocktäschel and Sebastian Riedel. 2017a. End-to-end differentiable proving. In *NIPS*.

Tim Rocktäschel and Sebastian Riedel. 2017b. End-to-end differentiable proving. In *NeurIPS*.

Brian H Ross. 1984. Remindings and their effects in learning a cognitive skill. *Cognitive psychology*.

Stéphane Ross, Geoffrey J Gordon, and Drew Bagnell. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*.

Apoorv Saxena, Aditay Tripathi, and Partha Talukdar. 2020. Improving multi-hop question answering over knowledge graphs using knowledge base embeddings. In *ACL*.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2008. The graph neural network model. *IEEE transactions on neural networks*.

Roger C Schank. 1982. *Dynamic memory: A theory of reminding and learning in computers and people*. cambridge university press.

Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks. In *ESWC*.

Henk Schmidt, Geoffrey Norman, and Henny Boshuizen. 1990. A cognitive perspective on medical expertise: theory and implications. *Academic medicine*.

Stefan Schoenmackers, Oren Etzioni, Daniel S Weld, and Jesse Davis. 2010. Learning first-order horn clauses from web text. In *EMNLP*.

Peter Shaw, Ming-Wei Chang, Panupong Pasupat, and Kristina Toutanova. 2020. Compositional generalization and natural language variation: Can a semantic parsing approach handle both? *arXiv preprint arXiv:2010.12725*.

Yelong Shen, Po-Sen Huang, Jianfeng Gao, and Weizhu Chen. 2017. Reasonet: Learning to stop reading in machine comprehension. In *KDD*.

Baoxu Shi and Tim Weninger. 2018. Open-world knowledge graph completion. In *AAAI*.

Livio Baldini Soares, Nicholas FitzGerald, Jeffrey Ling, and Tom Kwiatkowski. 2019. Matching the blanks: Distributional similarity for relation learning. In *ACL*.

Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. 2013. Reasoning with neural tensor networks for knowledge base completion. In *Neurips*.

Yu Su, Huan Sun, Brian Sadler, Mudhakar Srivatsa, Izzeddin Gür, Zenghui Yan, and Xifeng Yan. 2016. On generating characteristic-rich question sets for qa evaluation. In *EMNLP*.

Fabian Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2007. Yago: A core of semantic knowledge. In *WWW*.

Alane Suhr, Ming-Wei Chang, Peter Shaw, and Kenton Lee. 2020. Exploring unexplored generalization challenges for cross-database semantic parsing. In *ACL*.

Sainbayar Sukhbaatar, Jason Weston, and Rob Fergus. 2015. End-to-end memory networks. In *NIPS*.

Haitian Sun, Tania Bedrax-Weiss, and William W Cohen. 2019a. Pullnet: Open domain question answering with iterative retrieval on knowledge bases and text. In *EMNLP*.

Haitian Sun, Bhuwan Dhingra, Manzil Zaheer, Kathryn Mazaitis, Ruslan Salakhutdinov, and William W Cohen. 2018. Open domain question answering using early fusion of knowledge bases and text. In *EMNLP*.

Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. 2019b. Rotate: Knowledge graph embedding by relational rotation in complex space. In *ICLR*.

Ilya Sutskever, Joshua B Tenenbaum, and Russ R Salakhutdinov. 2009. Modelling relational data using bayesian clustered tensor factorization. In *Neurips*.

Alon Talmor and Jonathan Berant. 2018. The web as a knowledge-base for answering complex questions. In *NAACL-HLT*.

Jizhi Tang, Yansong Feng, and Dongyan Zhao. 2019. Learning to update knowledge graphs by reading news. In *EMNLP*.

Komal Teru, Etienne Denis, and Will Hamilton. 2020a. Inductive relation prediction by subgraph reasoning. In *ICML*.

Komal K Teru, Etienne Denis, and William L Hamilton. 2020b. Inductive relation prediction by subgraph reasoning. In *ICML*.

Kristina Toutanova and Danqi Chen. 2015. Observed versus latent features for knowledge base and text inference. In *Continuous Vector Space Models and their Compositionality Workshop*.

Kristina Toutanova, Danqi Chen, Patrick Pantel, Hoifung Poon, Pallavi Choudhury, and Michael Gamon. 2015. Representing text for joint embedding of text and knowledge bases. In *EMNLP*.

Kristina Toutanova, Victoria Lin, Wen-tau Yih, Hoifung Poon, and Chris Quirk. 2016. Compositional learning of embeddings for relation paths in knowledge base and text. In *ACL*.

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

Shikhar Vashishth, Prince Jain, and Partha Talukdar. 2018. Cesi: Canonicalizing open knowledge bases using embeddings and side information. In *WWW*.

Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha Talukdar. 2020. Composition-based multi-relational graph convolutional networks. In *ICLR*.

Petar Velickovic, Guillem Cucurull, A. Casanova, A. Romero, P. Liò, and Yoshua Bengio. 2018. Graph attention networks. In *ICLR*.

Pat Verga, Haitian Sun, Livio Baldini Soares, and William W Cohen. 2020. Facts as experts: Adaptable and interpretable neural memory over symbolic knowledge. *arXiv preprint arXiv:2007.00849*.

Patrick Verga, David Belanger, Emma Strubell, Benjamin Roth, and Andrew McCallum. 2016a. Multilingual relation extraction using compositional universal schema. In *NAACL*.

Patrick Verga, David Belanger, Emma Strubell, Benjamin Roth, and Andrew McCallum. 2016b. Multilingual relation extraction using compositional universal schema. In *NAACL*.

Hongwei Wang, Hongyu Ren, and Jure Leskovec. 2020. Entity context and relational paths for knowledge graph completion. *arXiv preprint arXiv:2002.06757*.

William Yang Wang, Kathryn Mazaitis, and William W Cohen. 2013. Programming with personalized pagerank: a locally groundable first-order probabilistic logic. In *CIKM*.

Robert West, Joelle Pineau, and Doina Precup. 2009. Wikispeedia: An online game for inferring semantic distances between concepts. In *IJCAI*.

Jason Weston, Emily Dinan, and Alexander Miller. 2018. Retrieve and refine: Improved sequence generation models for dialogue. In *ConvAI Workshop EMNLP*.

Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*.

Sam Wiseman and Karl Stratos. 2019. Label-agnostic sequence labeling by copying nearest neighbors. In *ACL*.

Peng Wu, Shujian Huang, Rongxiang Weng, Zaixiang Zheng, Jianbing Zhang, Xiaohui Yan, and Jiajun Chen. 2019. Learning representation mapping for relation detection in knowledge base question answering. In *ACL*.

Wenhan Xiong, Thien Hoang, and William Yang Wang. 2017. Deeppath: A reinforcement learning method for knowledge graph reasoning. In *EMNLP*.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How powerful are graph neural networks? In *ICLR*.

Zhao Xu, Volker Tresp, Kai Yu, and Hans-Peter Kriegel. 2006. Infinite hidden relational models. In *UAI*.

Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2015. Embedding entities and relations for learning and inference in knowledge bases. In *ICLR*.

Fan Yang, Zhilin Yang, and William W Cohen. 2017. Differentiable learning of logical rules for knowledge base reasoning. In *NeurIPS*.

Wen-tau Yih, Matthew Richardson, Christopher Meek, Ming-Wei Chang, and Jina Suh. 2016. The value of semantic parse labeling for knowledge base question answering. In *ACL*.

Dong Yu, Kaisheng Yao, Hang Su, Gang Li, and Frank Seide. 2013. Kl-divergence regularized deep neural network adaptation for improved large vocabulary speech recognition. In *ICASSP*.

Mo Yu, Wenpeng Yin, Kazi Saidul Hasan, Cicero dos Santos, Bing Xiang, and Bowen Zhou. 2017. Improved neural relation detection for knowledge base question answering. *arXiv preprint arXiv:1704.06194*.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *EMNLP*.

Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. 2020. Big bird: Transformers for longer sequences. In *Neurips*.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan Salakhutdinov, and Alexander Smola. 2017. Deep sets. In *NIPS*.

Wojciech Zaremba and Ilya Sutskever. 2015. Reinforcement learning neural turing machines. *arXiv:1505.00521*.

John M Zelle and Raymond J Mooney. 1996. Learning to parse database queries using inductive logic programming. In *NCAI*.

Daojian Zeng, Kang Liu, Siwei Lai, Guangyou Zhou, Jun Zhao, et al. 2014. Relation classification via convolutional deep neural network. In *COLING*.

Jingyi Zhang, Masao Utiyama, Eiichro Sumita, Graham Neubig, and Satoshi Nakamura. 2018a. Guiding neural machine translation with retrieved translation pieces. In *NAACL*.

Muhan Zhang and Yixin Chen. 2018. Link prediction based on graph neural networks. In *Neurips*.

Yuyu Zhang, Hanjun Dai, Zornitsa Kozareva, Alexander J Smola, and Le Song. 2018b. Variational reasoning for question answering with knowledge graph. In *AAAI*.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*.

Jun Zhu, Jiaming Song, and Bei Chen. 2012. Max-margin nonparametric latent feature models for link prediction. In *ICML*.