End-to-End Learning with Text & Knowledge Bases

Bhuwan Dhingra

CMU-LTI-20-002

May 13, 2020

Language Technologies Institute
School of Computer Science
Carnegie Mellon University
5000 Forbes Ave., Pittsburgh, PA 15213
www.lti.cs.cmu.edu

Thesis Committee:
William W. Cohen (co-chair)
Ruslan Salakhutdinov (co-chair)
Graham Neubig
Michael Collins (Columbia, Google)

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in Language and Information Technologies

© 2020, Bhuwan Dhingra

Keywords: natural language processing, machine learning, knowledge representation, knowledge bases, reading comprehension, question answering, deep learning

Dedicated to Prof Amitabha Mukherjee,
who inspired me to pursue a career in research.

Abstract

Deep learning has been tremendously successful on tasks where the output prediction depends on a small, relatively clean input, such as a single image or a passage of text. But for many tasks, such as answering factoid questions or fact verification, the output depends on a broader context, or background knowledge, which goes beyond the input. Despite the best efforts to automatically create large knowledge bases, in most domains a majority of the information is only available in raw unstructured text. Harnessing this knowledge requires fine-grained language understanding at the document level, and scalable reasoning procedures to aggregate information across documents. While neural networks excel at the former, it is not clear how to scale them for the latter. On the other hand, symbolic representations such as knowledge graphs support scalable reasoning algorithms, but these are difficult to integrate with gradient-based learning.

This thesis develops methods which leverage the strength of both neural and symbolic approaches. Specifically, we augment raw text with symbolic structure about entities and their relations from a knowledge graph, and learn task-specific neural embeddings of the combined data structure. We also develop algorithms for doing multi-step reasoning over the embeddings in a *differentiable* manner, leading to end-to-end models for answering complex queries. Along the way we develop variants of recurrent and graph neural networks suited to modeling textual and multi-relational data, respectively, and use transfer learning to improve generalization. Through a series of experiments on factoid question answering, task-oriented dialogue, language modeling, and relation extraction, we show that our proposed models perform complex reasoning over rich fine-grained information.

Acknowledgments

I am grateful to my advisers William Cohen and Russ Salakhutdinov. Their insight and knowledge, combined with the astonishing amount of trust they placed in me, is what made this research possible. William is a role model for the academic I aspire to be: brilliant, kind and funny. Beyond machine learning, he taught me the importance of taking it easy and having a sense of humor. Russ is one of the most brilliant researchers I have met and, at the same time, a compassionate mentor. His humility will continue to guide me well beyond my PhD.

The nice thing about studying at CMU is the number of world-class faculty you have the opportunity to learn from. I had the chance to collaborate with a few, including Zachary Lipton, Graham Neubig and Alan Black, who all contributed to my development as a researcher. I would like to thank all my teachers, in particular Roni Rosenfeld, Larry Wasserman and Ryan Tibshirani, whose styles have influenced my own teaching philosophy. I would like to thank Michael Collins for serving on my thesis committee and providing valuable feedback. I'm also grateful to Stacey Young, who provides tireless support to the students in the department, and always remains available to help.

None of the research in this thesis was carried out in a vacuum, and I would like to thank an exceptional group of collaborators: Zhilin Yang, Manzil Zaheer, Danish, Qiao Jin, Hanxiao Liu, Katie Mazaitis, Haitian Sun, Vidhisha Balachandran, Dheeraj Rajagopal, and Lidong Bing. I was also fortunate to have multiple industry internships and would like to acknowledge my mentors: Lihong Li, Jianfeng Gao, Yun-Nung Chen, Li Deng, George Dahl, Christopher Shallue, Mohammad Norouzi, Manaal Faruqui, Ankur Parikh and Dipanjan Das. Funding for this thesis came from NSF, Google, Apple, Microsoft, NVIDIA, and a PhD fellowship from Siemens.

My time in Pittsburgh was made significantly more enjoyable by the group of friends I made: Chaitanya, Sushil, Vikesh, Shrimai, Ankush, Devendra, Kanthashree, Bijit, Priyank, Rama, Leo and Bernie. I am indebted to my mom Rama, dad Krishan and sister Nivedita for their unwavering support and love in all my endeavours. Finally, I will always remember Pittsburgh as the place where I found my most precious prize, my wife Rolly, whose infinite love and patience keeps me going even in the toughest times.

Contents

1 Introduction **1**

1.1 Contributions . . . 3

1.2 Thesis Outline . . . 5

2 Background **7**

2.1 Prior Work . . . 7

2.1.1 Symbolic Knowledge Representation . . . 7

2.1.2 Knowledge Bases . . . 8

2.1.3 Factoid Question Answering . . . 10

2.2 Relevant Methods . . . 12

2.2.1 Representing Text . . . 12

2.2.2 Representing Graphs . . . 15

2.2.3 Retrieval . . . 16

I Learning to Read **19**

3 Reading Comprehension **21**

3.1 Overview . . . 22

3.2 Gated-Attention Reader . . . 23

3.3 Extending with Coreference . . . 27

3.4 Experiments . . . 30

3.4.1 Cloze-style QA . . . 30

3.4.2 Reasoning Tasks . . . 37

3.5 Related Work . . . 42

3.5.1 Neural Network Readers . . . 42

3.5.2 Linguistic Biases in Deep Learning . . . 43
3.6 Discussion . . . 44

4 Transfer Learning **47**
4.1 Overview . . . 47
4.2 Analysis of Word Embeddings . . . 49
4.2.1 Reading Comprehension Setup . . . 49
4.2.2 Pretraining Methods . . . 50
4.2.3 Performance Comparison . . . 51
4.2.4 Handling OOV tokens . . . 53
4.3 Cloze Pretraining . . . 54
4.3.1 System . . . 55
4.3.2 Experiments . . . 57
4.3.3 Analysis . . . 59
4.4 Discussion . . . 61

II Learning with Knowledge Graphs **63**

5 Open-Domain QA **65**
5.1 Overview . . . 65
5.2 Retrieval . . . 67
5.3 GRAFT-Nets . . . 69
5.4 Experiments & Results . . . 74
5.4.1 Datasets . . . 74
5.4.2 Main Results . . . 75
5.4.3 Analysis . . . 78
5.5 Discussion . . . 80

6 Multi-turn QA **81**
6.1 Overview . . . 81
6.2 Probabilistic KB Lookup . . . 83
6.3 KB-InfoBot . . . 85
6.4 End-to-End Training . . . 88
6.5 Experiments & Results . . . 90

6.5.1 Models & Data . . . 90
6.5.2 Simulated User Evaluation . . . 92
6.5.3 Human Evaluation . . . 94
6.6 Related Work . . . 95
6.7 Discussion . . . 96

III Text as a Virtual Knowledge Base 99

7 Lazy Slot-Filling 101
7.1 Overview . . . 101
7.2 Virtual Knowledge Base . . . 103
7.2.1 Preliminaries . . . 103
7.2.2 Dual Encoder . . . 104
7.2.3 Training & Inference . . . 105
7.2.4 Generalizing to Unseen Relations . . . 106
7.3 Experiments & Results . . . 107
7.3.1 Setup . . . 107
7.3.2 Generalization to Unseen Entities . . . 109
7.3.3 Generalization to Unseen Relations . . . 111
7.3.4 Further Analysis . . . 112
7.4 Discussion . . . 113

8 Differentiable Reasoning 117
8.1 Overview . . . 117
8.2 Differentiable Reasoning over a KB of Indexed Text . . . 119
8.2.1 Differentiable Multi-Hop Reasoning . . . 120
8.2.2 Efficient Implementation . . . 122
8.2.3 Pretraining . . . 125
8.3 Experiments . . . 126
8.3.1 METAQA: Multi-Hop Question Answering with Text . . . 126
8.3.2 WikiData: Multi-Hop Slot-Filling . . . 129
8.3.3 HotpotQA: Multi-Hop Information Retrieval . . . 132
8.3.4 HotpotQA: End-to-End Answer Extraction . . . 134
8.4 Discussion . . . 135

9 Conclusions **137**

9.1 Summary of Contributions 137

9.2 Key Ideas 139

9.3 Future Work 140

Bibliography **145**

List of Figures

2.1 (Left) A binary fact and (Right) an N-ary fact represented in graphical form in a KB. The N-ary fact introduces a dummy node to connect the different arguments of the relation POSITION-HELD. 9

3.1 Example questions which require coreference-based reasoning from the bAbi dataset [227] (top) and Wikihop dataset [223] (bottom). Coreferences are in bold, and the correct answers are underlined. 23
3.2 Gated-Attention Reader. Dashed lines represent dropout connections. 24
3.3 Forward (left) and Backward (right) Coref-RNN layers. *Mary* and *she* are coreferent. 27
3.4 Alternative view of Coref-RNNs as a memory network. Each memory cell corresponds to a coreference cluster, and read / write operations are performed on it when a token from that cluster is encountered in the text. 29
3.5 Performance in accuracy with and without the Gated-Attention module over different training sizes. p-values for an exact one-sided Mcnemar's test are given inside the parentheses for each setting. 34
3.6 Layer-wise attention visualization of GA Reader trained on WDW-Strict. 36
3.7 (Left) Accuracy of GA w/ C-GRU as coreference annotations are removed for bAbi task 3. (Right) Expected probability of correct answer ($\exp(-loss)$) on Validation set as training progresses on Wikihop dataset for 1K, 5K and the full training datasets. 40

4.1 Test set accuracies and std error on the Who-Did-What dataset for Stanford AR and GA Reader, trained after initializing with word vectors induced from different corpora. Without controlling for the initialization method, different conclusions may be drawn about which architecture is superior. Corpus 1: BookTest dataset [8], Corpus 2: Wikipedia + Gigaword. 48

4.2 Test set accuracy and std error for GA Reader and Stanford AR on WDW (left, middle-left) and CBT-NE (middle-right, right) when trained after initializing with pre-trained embeddings induced from different corpora (Table 4.1), or randomly . . . 51
4.3 Test set accuracy and std error on (left) WDW when initialized with off-the-shelf GloVe embeddings of different sizes, (right) CBT-NE when initialized with embeddings trained on BT corpus after removing a fraction of stopwords (red), or using different window sizes (green). . . . 52
4.4 Test set accuracy and std error for GA Reader on WDW (left) and CBT-NE (right) when trained after assigning different cuts of the vocabulary with word vectors. *Min frequency* refers to the minimum count of a word type for it to be included in the vocabulary. . . . 53
4.5 Descriptions of the features extracted from the questions. . . . 59
4.6 Regression coefficients, along with std-errors, when predicting F1 score of *cloze* model, or *sl* model, or the difference of the two, from features computed from SQuAD dev set questions. . . . 60
4.7 Performance gain with pretraining for different subsets of question types. . . . 61

5.1 (Left) To answer a question posed in natural language, GRAFT-Net considers a heterogeneous graph constructed from text and KB facts, and thus can leverage the rich relational structure between the two information sources. (Right) Embeddings are propagated in the graph for a fixed number of layers (L) and the final node representations are used to classify answers. . . . 66
5.2 Illustration of the heterogeneous update rules for entities (Left) and text documents (Right). . . . 70
5.3 Directed propagation of embeddings in GRAFT-Net. A scalar *PageRank* score $pr_v^{(l)}$ is maintained for each node v across layers, which spreads out from the seed node. Embeddings are only propagated from nodes with $pr_v^{(l)} > 0$. . . . 72
5.4 Left: Effect of directed propagation and query-based attention over relations for the WebQuestionsSP dataset with 30% KB and 100% KB. Right: Hits@1 with different rates of fact-dropout on and WikiMovies and WebQuestionsSP. . . . 79

6.1 An interaction between a user looking for a movie and the KB-InfoBot. An entity-centric knowledge base is shown above the KB-InfoBot (missing values denoted by X). . . . 82

6.2 High-level overview of the end-to-end KB-InfoBot. Components with trainable parameters are highlighted in gray. 86

6.3 Performance of KB-InfoBot versions when tested against humans. (Left) Success rate, with the number of test dialogues indicated on each bar, and the p-values from a two-sided permutation test. (Right) Distribution of the number of turns in each dialogue (differences in mean are significant with $p < 0.01$). 94

6.4 Sample dialogues between users and the KB-InfoBot (RL-Soft version). Each turn begins with a user utterance followed by the *agent response*. Rank denotes the rank of the target movie in the KB-posterior after each turn. 95

7.1 The TARDIS model for answering queries by extracting a span from a given sentence. [CLS] is a special token appended to the beginning of each sequence. We train the model to predict [CLS] when the sentence does not contain the answer. The double arrow indicates that the parameters of BERT are shared between the sentence and query. For the probing experiments (§7.3.4), the BERT model (shaded red boxes) parameters are kept fixed. 102

7.2 Restricted setting: Micro-averaged F1 score on unseen relations, averaged across 10 folds, as the number of tail type seeds provided for each relation increases. For each relation, we repeat the experiment for 3 different randomly selected sets of tail type seeds and show the average and standard deviation (shaded blue region) here. 111

8.1 DrKIT answers multi-hop questions by iteratively mapping an input set of entities X (*The Grateful Dead*, *Bob Dylan*) to an output set of entities Y (*Dylan & the Dead*, *American beauty*, ...) which are related to any input entity by some relation R (*album by*). . . 119

8.2 Runtime on a single K80 GPU when using ragged representations for implementing sparse-matrix vector product, vs the default sparse-matrix times dense vector product available in TensorFlow. $|\mathcal{N}| > 10^5$ leads to OOM for the latter. 123

8.3 Hits @1 vs Queries/sec during inference on MetaQA (left) and WikiData (middle) tasks, measured on a single CPU server with 6 cores. MSR: Multi-step Retriever model from Das et al. [46] (we only show Q/sec). 127

8.4 Effect of varying number of nearest neighbors K during MIPS on the Hits@1 performance. 128

8.5 Macro-avg accuracy on lazy slot-filling. We split the results based on frequency of the relations in our WikiData training data. DrKIT-all spans refers to a variant of our model which selects from all spans in the corpus, instead of only entity-linked mentions. 131

List of Tables

3.1 Cloze-style QA dataset statistics. 30

3.2 Hyperparameter settings for each dataset. dim() indicates hidden state size of GRU. 31

3.3 Validation/Test accuracy $(\%)$ on CNN, Daily Mail and CBT. Results marked with "†" are cf previously published works. Results marked with "‡" were obtained by training on a larger training set. Best performance on standard training sets is in bold, and on larger training sets in italics. 33

3.4 Validation/Test accuracy (%) on WDW dataset for both "Strict" and "Relaxed" settings. Results with "†" are cf previously published works. 34

3.5 Performance of several model variants on WDW dataset without using qe-comm feature and with fixed $L(w)$. (Left) Gating functions in the GA module. (Middle) Number of hops K. (Right) Model variants. 35

3.6 Accuracy on bAbi-1K, averaged across all 20 tasks. Following previous work we run each task for 10 random seeds, and report the Avg and Max (based on the dev set). A task is considered failed if its Max performance is < 0.95. 37

3.7 Breakdown of task-wise performance on bAbi dataset. Tasks where C-GRU is significant better / worse than either GRU or QRNs are highlighted. 39

3.8 Accuracy on Wikihop for different number of training examples. *Follow:* subset annotated as answer follows from the given passages. *Follow + multiple:* subset annotated as requiring multiple passages for answering. *Follow + single:* subset annotated as requiring one passage for answering. $^{\dagger}p = 0.057$ using Mcnemar's test compared to GA w/ GRU. 41

3.9 Accuracy on LAMBADA test set, averaged across two runs with random initializations. *context:* passages for which the answer is in context. *overall:* full test set for comparison to prior work. $^{\dagger}p < 0.0001$ using Mcnemar's test compared to GA w/ GRU. 42

4.1 Details of corpora used for training word embeddings. OTS: Off-The-Shelf embeddings provided with GloVe / word2vec. Corpus size is in # of tokens. 50
4.2 An example constructed cloze. 55
4.3 A holistic view of the performance of our system compared against baseline systems on SQuAD and TriviaQA. Column groups represent different fractions of the training set used for training. 58
4.4 5-fold cross-validation results on BioASQ Task 5b. *Our SL experiments showed better performance than what was reported in Wiese et al. [230]. 58

5.1 Statistics of all the retrieved subgraphs $\cup_q \mathcal{G}_q$ for WikiMovies-10K and WebQuestionsSP. 74
5.2 Hits@1 / F1 scores of GRAFT-Nets (GN) compared to KV-MemNN (KV) in KB only (-KB), early fusion (-EF), and late fusion (-LF) settings. 75
5.3 Hits@1 / F1 scores compared to SOTA models using only KB or text: MINERVA [42], R2-AsV [221], Neural Symbolic Machines (NSM) [129], DrQA [26], R-GCN [177] and KV-MemNN [135]. *DrQA is pretrained on SQuAD. #Re-implemented. 77
5.4 Examples from WebQuestionsSP dataset. Top: The model misses a correct answer. Bottom: The model predicts an extra incorrect answer. 78
5.5 Non-Heterogeneous (NH) vs. Heterogeneous (H) updates on WebQuestionsSP . 79

6.1 Movies-KB statistics for four splits. Refer to Section 6.2 for description of columns. 92
6.2 Performance comparison. Average ($\pm$std error) for 5000 runs after choosing the best model during training. T: Average number of turns. S: Success rate. R: Average reward. 93

7.1 A slot-filling query with a distantly supervised positive sentence and a shared-entity and shared-relation negative each. 105
7.2 Restricted setting: Micro-averaged precision, recall and F1 on a held-out set of queries about unseen entities over unseen documents. All models are trained using only shared-entity negatives (§7.2.3). 109
7.3 Lazy slot-filling: micro- and macro-averaged (over relation types) accuracies of lazy slot-filling for TARDIS trained on entity negatives (*E-Neg*), random negatives (*Random-Neg*), and both entity and relation negatives (*E+R-Neg*). 109

7.4 Restricted setting: Micro-averaged precision, recall and F1 on unseen relations and unseen entities. The results are averaged across 10 folds, where in each fold a different subset of relations are held out for testing. The TARDIS models are trained only on shared-entity negatives. *+ Tail type seeds* denotes the zero-shot setup described in §7.2.4. 110

7.5 Lazy slot-filling: Micro- and Macro-averaged accuracies on unseen relations, using the TARDIS model. All models are trained on both negative types (*E+R-Neg*). We exclude any query whose answer is among the tail type seeds from the evaluation. 111

7.6 Probing experiments: Micro-averaged precision, recall and F1 for pre-trained BERT (uncased BERT-Base) vs Fine-tuned BERT (trained with the TARDIS model using entity and relation negatives *E+R-Neg*). 112

7.7 Lazy slot-filling predictions about unseen relations from the dual encoder model. We show the retrieved sentence and answer (in bold) from the NL Relation model trained on E+R-Neg, and the one which additionally uses 3 tail type seeds. 115

8.1 MetaQA (left) and WikiData (right) Hits @1 for 1-3 hop sub-tasks. ots: off-the-shelf without re-training. †: obtained from Sun et al. [197]. cascade: adapted to multi-hop setting by repeatedly applying Eq. 8.3. pre: pre-trained on slot-filling. e2e: end-to-end trained on single-hop and multi-hop queries. 127

8.2 Ablation study for different components of DrKIT. 128

8.3 WikiData multi-hop slot-filling dataset 129

8.4 Left: Retrieval performance on the HotpotQA benchmark dev set. Q/s denotes the number of queries per second during inference on a single 16-core CPU. Accuracy @k is the fraction where *both* the correct passages are retrieved in the top k. †: Baselines obtained from Das et al. [47]. For DrKIT, we report the performance when the index is pretrained using the WikiData KB alone, the HotpotQA training questions alone, or using both. *: Measured on different machines with similar specs. Right: Overall performance on the HotpotQA task, when passing 10 retrieved passages to a downstream reading comprehension model [246]. ‡: From Das et al. [47]. ◇: From Qi et al. [159]. †: Results on the dev set. 132

8.5 Official leaderboard evaluation on the test set of HotpotQA. #Bert refers to the number of calls to BERT [51] in the model. s/Q denotes seconds per query (using batch size 1) for inference on a single 16-core CPU. Answer, Sup Fact and Joint are the official evaluation metrics for HotpotQA. *: This is the minimum number of BERT calls based on model and hyperparameter descriptions in the respective papers. †: Computed using code released by authors, using a batch size of 1. ‡: Estimated based on the number of BERT calls, using 0.8s as the time for one call (without considering overhead due to other computation in the model). ◇: One call to a 5-layer Transformer, and one call to BERT. 134

Chapter 1

Introduction

> The driver of the power of intelligent systems is the knowledge the systems have about their universe of discourse, not the sophistication of the reasoning process the systems employ.
>
> – *Edward Feigenbaum* [37]

The goal of Artificial Intelligence (AI) is to build systems which can reason about their surroundings, but reasoning is only possible if the system has a representation of those surroundings. A representation, in this sense, is a substitute for the real world; an abstract thing which enables a person or a program to determine consequences internally by thinking, rather than externally by acting [49]. Hence, intelligent systems which aim to operate in an open environment, perform complex tasks, and engage with people in natural conversations, must store real-world information in a data structure which is amenable to algorithmic manipulation. Such a data structure is called a *knowledge representation*.

In this thesis we are interested in intelligent systems which use knowledge representations derived from large collections of data to answer user queries. Much of collective human knowledge resides on the internet today, in unstructured and semi-structured forms such as text, images, tables and lists. While half the world's population has access to the internet, and consequently this knowledge, no one can reasonably navigate it without the help of tools such as search engines and question answering systems. Internally, these tools must convert the underlying data to a representation which allows retrieving information based on the *semantics* of a query. Further, since the data may be incomplete, i.e. not all information may be written explicitly, we would also like this representation to support different forms of reasoning. In this case reasoning is the ability to derive facts which are implied from the facts which are known. So if our data contains the facts that "Barack Obama graduated from Columbia University in

1983", and "Columbia University is located in New York", the system should be able to reason that the answer to "Where did Barack Obama live in 1982?" is probably "New York".

Data structures which organize information in a structured format are commonly known as *Knowledge Bases (KBs)*. Among these, the most widely used format today is a knowledge graph [18, 188]. The nodes in this graph are real-world concepts (or entities), and typed edges represent relationships between those entities. The edge types are drawn from a fixed vocabulary, also known as predicates, and each instance of a relationship is called a fact. Graphs are an attractive data structure since reasoning (or inference) over their contents can often be modeled as a path-finding problem. So we can answer an information-seeking query by identifying the entities it mentions (*entity linking*) [70, 167], followed by the relations it asks for and any other constraints it specifies (*semantic parsing*) [15, 129, 250]. The entities, relations and constraints together specify a path which leads to the answer. Searching for this path can be done efficiently using well studied graph algorithms. Knowledge graphs have recently dominated industry applications—e.g. Google's graph[1] organizes information relevant to search queries, and the goal of the Semantic Web effort [16] is to organize the internet in a similar manner. In this thesis, as is often done in the NLP community, we will use the more general term *KBs* to refer to the specific format of a knowledge graph.

There are, however, some limitations with this type of knowledge representation, which form the focus of this thesis. (1) To allow accurate semantic parsing, KBs usually limit the number of allowed predicates they contain. Consequently, the vast majority of information in a domain cannot be encoded in the KB.[2] (2) The information extraction pipelines which are used to populate KBs typically favor precision over recall. Hence, even facts which can be encoded in a KB are often missing [136]. (3) The world changes with time, and so do its entities and their relationships. Detecting these changes for updating a KB involves a delay. (4) Reasoning algorithms over KBs are generally *symbolic*, and hence difficult to integrate with gradient-based deep learning. This makes it difficult to build systems which learn to reason over the contents of a KB using only downstream task data.

To address these limitations, we develop methods which *directly treat a text corpus as a knowledge base*. Specifically, we answer queries by extracting spans from text and reasoning over them on-the-fly. People typically share information via text, be it in the form of news

[1] `https://developers.google.com/knowledge-graph`

[2] One example of this can be observed by comparing the Wikipedia article of an entity with its corresponding WikiData page. The latter is a structured KB and only encodes up to a dozen facts about even the most popular entities, whereas Wikipedia articles may contain hundreds of facts.

reports, blog posts, scientific articles, online forums or encyclopedia entries. Hence, in most domains it is relatively easy to collect a text corpus providing richer and more up-to-date information than a structured knowledge base. The challenge lies in understanding natural language to find answers to queries, since now instead of a closed set of predicates we must deal with an open set of textual patterns. The approaches we present here leverage advances in neural language modeling [13] to perform this complex task using labeled question and answer pairs. The underlying knowledge representation consists of learned feature vectors for all *spans* in the corpus, in some cases accompanied with extra symbolic structures like sparse inverted indexes. We also discuss neural modules which implement reasoning procedures over these representations, and can be trained *end-to-end* using the labeled data. Importantly, instead of replacing KBs altogether, we develop techniques which leverage their structure to improve generalization and reasoning over text.

Concretely, this thesis attempts to answer the following research questions:

1. How should we map text spans and their contexts to a representation of their meaning? Specifically, which neural architectures learn representations efficiently from data? And, do linguistically motivated inductive biases improve generalization?
2. How can we model dependencies between multiple passages and documents? Can we incorporate a graph structure similar to KBs for utilizing long-range context in the representation learning process? Can this be made scalable?
3. How can we combine representation learning with symbolic reasoning to answer complex queries? Are there differentiable implementations for the latter which allow end-to-end optimization of the former?

1.1 Contributions

In this section we outline the core contributions of this thesis.

Reading Comprehension (Chapters 3 & 4). The standard pipeline for answering queries against a corpus proceeds in two steps—*retrieving* relevant passages likely to contain the answer, followed by *reading* those passages to extract the answer [26, 55, 221]. Models designed for the latter task must deal with complex linguistic phenomena such as paraphrasing, coreference, logical entailment, syntactic and semantic dependencies, and so on [24]. We present a neural network architecture, called *Gated-Attention (GA) Reader*, for this task which employs

a novel attention mechanism [7] to construct query-dependent representations of the passage. We show that such representations outperform query-independent ones, as well as those obtained via other forms of attention. Recurrent Neural Networks (RNNs) are widely used for modeling text sequences but have difficulty capturing long-term dependencies in the input [12], such as those when there are coreferent mentions in the input text. Hence, we develop *Coref-RNNs*, which explicitly model coreferent mentions detected by an external system. To improve the efficiency of these models, we also develop a simple yet highly effective semi-supervised learning technique which leverages unlabeled semi-structured documents. We show that these methods show strong performance on many standard benchmarks for language understanding. The content of these chapters appeared in the proceedings of ACL 2017 [54], and as two short papers in NAACL 2018 [56, 57].

Joint Representations of KBs and Text (Chapter 5). Retrieving and reading text is a flexible approach for answering the bulk of open-domain questions, but can be brittle when presented with new patterns at test time [102]. It also cannot handle complex questions whose answers need to be aggregated from many documents, since these may not be retrieved in the first place [200]. Hence we develop a model, called *Graft-Net*, for extracting answers from a combined graph data structure consisting of text linked to a structured KB via *entity* mentions [63]. In one direction, the KB provides background knowledge for the concepts mentioned in the text, and in the other direction, the text adds missing edges between entities in the KB. We show that this combination improves over both text-only and KB-only methods, while being applicable across a wide range of KB completeness settings. This chapter first appeared in the proceedings of EMNLP 2018 [196].

Multi-Step Retrieval over KBs (Chapter 6). For complex information needs, users may not be able to form a complete well-formed query to the system in a single turn. Instead, a multi-turn setting makes more sense, where the user starts with an under-specified query, and the system responds with follow-up questions to locate the information requested. We explore this problem when the underlying knowledge source is an entity-centric KB, i.e. facts in the KB specify attributes of a list of entities. We introduce a probabilistic framework for computing the posterior distribution over the contents of the KB, given belief states derived from the user inputs. We use this framework to build a modular dialogue system and train it *end-to-end* using reinforcement learning over feedback provided by the users. We experiment with both simulated and real users at the task of retrieving movies from a movie database. This chapter

first appeared in the proceedings of ACL 2017 [53].

Text as a Virtual Knowledge Base (Chapters 7 & 8). Recent advances in contextual representation learning [51, 156, 247] suggest a new paradigm for interacting with textual knowledge. Instead of retrieving and reading a few passages given a query, we create an offline index of contextual embeddings for all important spans in the corpus, which we call a *Virtual Knowledge Base*. Instead of entities, this KB stores information at a mention level, with a higher recall since the embeddings encode an open class of textual relations. Given a query, we leverage efficient algorithms for maximum inner product search (MIPS) [1, 103, 181] to retrieve relevant spans from the KB. We also develop a differentiable operator for traversing paths of relations in the embedding space, and show its application to multi-hop question answering [246]. Part of this work first appeared in the proceedings of ICLR 2020 [59].

1.2 Thesis Outline

We begin by giving an overview of prior work in Chapter 2, and outline some general techniques used throughout the thesis. The next 6 chapters (3-8) are divided into three parts. The first part (3 and 4) explores machine reading—the task of understanding language—through the lens of question answering over small contexts. The second part (5 and 6) shifts the focus towards structured KBs, and explores how these can be combined with text for question answering and dialogue. The last part (7 and 8) introduces Virtual Knowledge Bases derived from contextual representations of text, and discusses methods for pre-training and extracting answers from them. Concluding remarks and a discussion of future research directions is included in Chapter 9.

Chapter 2

Background

This chapter is divided into two parts. Section 2.1 situates this thesis in the context of prior work, and section 2.2 introduces some relevant techniques and algorithms which are used in subsequent chapters.

2.1 Prior Work

We start with a brief history of knowledge representation (§ 2.1.1) and highlight its close connections to First Order Logic (FOL). We then discuss methods for constructing and reasoning over large-scale knowledge bases (§ 2.1.2). This is followed by an overview of work which has explored answering open-domain factoid questions (§ 2.1.3).

2.1.1 Symbolic Knowledge Representation

Formally, Knowledge Representation (KR) is the study of meaning and how it can be expressed in a language such that we can compute with it. Often, the computation that we are interested in is *inference*—deriving what is true based on what is known. Hence, a natural choice for the language of knowledge is First Order Logic (FOL), with its sophisticated machinery for reasoning and inference. Indeed, much of the history of KR has been concerned with building programs implementing some subset of FOL suited for a particular application. We refer the reader to Brachman and Levesque [22] for a detailed description of several such variants.

The earliest example of automated reasoning over a formal language is the pioneering work of Alan Newell and Herbert Simon on the Logic Theorist [148] and the General Problem Solver [149]. These were the first programs to separate the knowledge of a system from the procedures

for using it, notably deriving proofs for 38 theorems from the *Principia Mathematica* and solving problems like the Tower of Hanoi.[1] Techniques for general problem solving, however, had limited success on complex real-world problems. Hence, the 60s and 70s saw increased effort on building *Expert Systems*—programs engineered to solve specific problems in a domain [99]. Their high-level design, consisting of a knowledge base encoding facts and rules about the domain and an inference engine for deriving new facts from them, continues to influence much of the work in AI today [82]. MYCIN was one such system for diagnosing medical conditions [186], though it was never used in practice. Around this time similar ideas were also explored for natural language understanding, e.g. in the system SHRDLU [234].

An alternate line of work has looked at representing knowledge using *frames* [140]. A frame is an abstract structured type, with slots whose values relate it to other frames, and methods which describe how it behaves in different contexts. Frames are closely related to classes in object-oriented programming, and their representation power can be described in terms of logic [90, 108]. Frame semantics [72] applies the same ideas to language understanding by constructing frames associated with words [9].

Levesque and Brachman [120] pointed out a fundamental limitation with symbolic knowledge representations—the more expressive a language is, in terms of the kinds of facts it can express, the harder it is to do automated reasoning with it, in terms of the tractability of computation. One of the goals of this thesis is to explore whether we can overcome this limitation via *approximate reasoning*, where the answer returned is correct only with a probability. Hence, we consider the most expressive language for representing knowledge we know of—natural language—and develop neural representations and reasoning procedures for answering questions over it. While our focus is entirely on English, the methods we present should be more generally applicable to other natural languages as well.

2.1.2 Knowledge Bases

This thesis is concerned with storing encyclopedic knowledge—relatively unambiguous facts about people, places and things in domains such as science, politics, geography, history and entertainment. The most common format for storing this type of information is a graph-structured knowledge base. We will concern ourselves with the simplest form of such a KB, denoted as $\mathcal{K} = (\mathcal{N}, \mathcal{E}, \mathcal{R})$, where the nodes $\mathcal{N}$ represent the set of entities in the KB, and the edges $\mathcal{E}$ are triples (s, r, o) which denote that relation $r \in \mathcal{R}$ holds between the subject $s \in \mathcal{N}$ and object

[1] `https://en.wikipedia.org/wiki/TowerofHanoi`

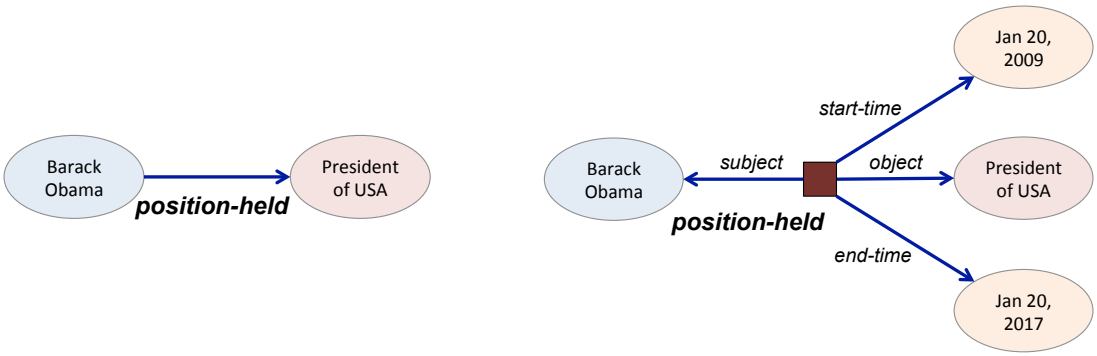


Figure 2.1: (Left) A binary fact and (Right) an N-ary fact represented in graphical form in a KB. The N-ary fact introduces a dummy node to connect the different arguments of the relation POSITION-HELD.

$o \in \mathcal{N}$. Borrowing terminology from FOL, the relations in $\mathcal{R}$ are also called predicates, and the triples in $\mathcal{E}$ are also called facts. Together, $\mathcal{R}$ and $\mathcal{N}$ denote the *schema* of the KB, and define a strict limit on the information which can be populated in the KB.

Note that the above formulation only allows for *binary* predicates. A workaround for populating higher order predicates is to introduce dummy nodes which connect the different arguments of a predicate.[2] An example is shown in Figure 2.1, and we use this workaround in Chapter 5. Some entities in the KB may correspond to types, such as PERSON and ORGANIZATION, and instances of a type can be connected to it via the Is-A relation. Types can be organized into hierarchies using relations such as SUBCLASS-OF. Several large KBs have been constructed to hold general knowledge through a mix of collaborative community effort and automatic relation extraction pipelines [142]. In the research community, some widely used KBs of this sort are Freebase [18], Wikidata [212] and NELL [141].

Knowledge Base Completion (KBC). Despite the best efforts, however, these KBs remain highly incomplete, i.e. most entities have only a few predicates populated [136]. In many cases these missing facts can be inferred from other facts which are known about the entity, e.g. if we know (MELINDA GATES, FOUNDER, GATES FOUNDATION) and (GATES FOUNDATION, HEADQUARTERS, SEATTLE), then we can reasonably infer that (MELINDA GATES, WORKS-IN, SEATTLE). This task is called Knowledge Base Completion, and has been the focus of many approaches over the years. These include classical statistical relational learning approaches [75, 113, 117] and their recent neural counterparts [161, 162], compositional vector space models [43, 85, 147], and learning interpretable rules via reinforcement learning [45, 131, 185, 239]. Recently, an intrigu-

[2]In Freebase [18] these are known as *Compound Value Types (CVTs)*.

ing line of work has looked at "neuralizing" symbolic operations like relation following [35, 36] and backward chaining [139, 171] for doing inference over KBs. In Chapter 8 we will extend this last line of work to do reasoning over text.

Joint Representations of KBs & Text. The vast majority of facts missing from a KB may not be inferrable from other facts present in the KB. Further, many interesting facts may not be encodable using the predicates allowed by the schema of the KB, and in fact much of human knowledge is intrinsically hard to express in the form of tuples, such as common sense and procedural knowledge. Unstructured text offers a rich complement for these limitations, and joint representations of KBs and text have been explored for relation extraction and KBC [41, 88, 118, 169, 205, 210]. The key idea behind these works is to represent both text sentences and KB facts in a shared feature space which captures an open class of predicates, termed *Universal Schema*. In Chapter 5, we will introduce a novel model based on Graph Convolution Networks, which allows learning richer representations by exchanging information between text and facts in a query-dependent manner.

2.1.3 Factoid Question Answering

A fundamental issue in learning with knowledge is the ability to automatically retrieve relevant knowledge from a large source, as required by the task at hand. This is potentially useful for many complex AI systems—detecting fake news, automating scientific discoveries, medical diagnosis etc.—but these rely on several other capabilities well. Instead, in this thesis we will use factoid question answering (QA) as benchmark for measuring progress towards learning with knowledge. Given a natural language query q and a knowledge source $\mathcal{G}$, which can be a KB or a text corpus or both, our goal will be to extract a natural language answer a. In some cases, we will consider semi-structured queries which are not strictly natural language, but nevertheless have a string representation (e.g. (Melinda Gates, Works-In, ?) can be represented as "Melinda Gates . Works In ?"). We will also consider cases where the query has multiple ground truth answers $A = \{a_1, a_2, \ldots\}$ (e.g. "Which Fortune 500 companies are headquartered in Seattle?").

The term *factoid* simply means that the questions are about facts, or assertions which are true given the knowledge source. In the case of a KB, it is usually assumed that some subset of all the possible facts $\mathcal{T} \subset \mathcal{N} \times \mathcal{N} \times \mathcal{R}$ is true, and the observed facts are a subset of all the true facts. Hence, any fact which is not in the KB may be true or false, but every fact in

the KB is definitely true. For text corpora, this is more difficult, since text assertions may be open to interpretation, and sometimes different assertions in the same corpus may contradict each other. While this is an important issue for text-based QA systems, we will side-step it here and instead assume that each question has exactly one unambiguous set of answers which are spans in the corpus. Defined in this manner, systems can be easily evaluated on their QA ability by measuring the accuracy of their answers. Collecting data for these type of questions is also much easier than those with longer subjective answers. The methods presented in this thesis all assume a *supervised* setting, where we have access to a (potentially large) collection of QA pairs for training, however in Chapter 4 we will also show how unlabeled text can be used in the process.

Knowledge-Based Question Answering. Traditional approaches for answering questions against a knowledge base rely on two components – (1) a *semantic parser* which maps questions to a logical form, e.g. $\lambda x.\exists y.\textsc{Founder}(\textsc{Melinda Gates},y).\textsc{Headquarters}(y,x)$; and (2) a *query language* for executing the logical form against the KB to compute the answer [14, 114, 168, 258, 259]. A challenging setting in KB-QA is learning from denotations, where the logical forms are latent during training and must be inferred from question answer pairs [130]. Recently, deep learning has also been applied to KB-QA. For simple questions where the logical forms contain only a single predicate, memory networks have shown strong performance [21, 100]. For complex questions where the logical forms involve reasoning over multiple predicates, reinforcement learning has been used [42, 129]. To deal with incomplete KBs, Gardner and Krishnamurthy [73] additionally use features extracted from text and an open-vocabulary semantic parser, and Ryu et al. [172] use a pipelined system for aggregating evidence from both unstructured and semi-structured sources. Das et al. [44] encode both KB facts and text sentences into a common embedding space [169] and use Key-Value Memory Networks (KV-MemNNs) for reasoning over it [135]. In Chapter 5 we will show that this can be improved by using Graph Convolution Networks (GCNs) instead [110].

Text-Based Question Answering. Using text corpora as a knowledge source for answering open-domain questions goes back to the QA track evaluations of the Text REtrieval Conference (TREC) [211]. Early systems used a pipeline consisting of question parsing, answer type determination, document retrieval, answer candidate generation, and answer reranking [143]. Many of these components were further refined for the famous IBM Watson system which beat human participants on the *Jeopardy!* quiz show [71]. However, such heavily engineered pipelines

are difficult to replicate outside the domain in which they are developed.[3] Recent years have seen the application of deep learning to this task, reducing the number of components in the pipeline to two – (1) a *text retriever* for identifying passages of text relevant to the question, and (2) a *reading comprehension* model to extract the answer span from the relevant passages [26, 164, 217, 218, 221]. The retriever is typically implemented using a fast and shallow method, such as TFIDF, while complex neural network models have been developed for reading comprehension [81, 98, 179, 184, 255]. The latter rely on large-scale training datasets such as SQuAD [166], TriviaQA [104] and Natural Questions [115]. In Chapters 3 and 4 we will present one such model and methods for training it.

2.2 Relevant Methods

In this section we outline the common tools and techniques used throughout the thesis, including neural models for contextually representing text (§ 2.2.1) and nodes in a graph (§ 2.2.2), as well as shallow retrieval methods (§ 2.2.3).

2.2.1 Representing Text

The power of neural networks lies in learning features of data given labels for a particular task. For textual data, we will assume the data is in the form of a sequence of words $s = w_1, w_2, \ldots, w_N$, where each w_i comes from a fixed vocabulary $\mathcal{V}$ and can be represented as a one-hot vector, i.e. $w_i \in \{0,1\}^{\mathcal{V}}$. Depending on the application, we will either represent the entire sequence as a single fixed size vector $h_s \in \mathbb{R}^d$, or as a sequence of vectors $h_1, h_2, \ldots, h_N$, s.t. $h_i \in \mathbb{R}^d$, where d is the embedding size. In the latter case, we are interested in contextual representations, i.e. h_i should be a function of the meaning of w_i in the context of s. Below we discuss two widely used parametric models for deriving these vectors.

Recurrent Neural Networks (RNNs). An RNN computes the contextual representations of tokens by initializing a hidden vector h_0 and recursively computing:

$$h_i = f(w_i, h_{i-1}), \quad \forall i = 1, \ldots, N. \tag{2.1}$$

Note that a single RNN only captures left-sided context. In order to capture right-sided context in the representation as well, we can use bi-directional RNNs, by initializing both h_0 and h_{N+1},

[3] Watson required "3 years of intense research and development by a core team of about 20 researchers" [71].

and computing:

$$h_i^l = f^l(w_i, h_{i-1}), \quad h_i^r = f^r(w_i, h_{i+1}), \quad h_i = [h_i^l; h_i^r], \quad \forall i = 1, \ldots, N.$$

Here we concatenate h_i^l and h_i^r to obtain the final representation for w_i. The functions f^l, f^r are neural networks with trainable parameters—in the simplest case feedforward layers [66]. Such "vanilla" RNNs have difficulties in learning dependencies between tokens far apart in the input sequence when trained using error backpropagation [12], since gradients may either explode or vanish when repeatedly multipled across time-steps. Two variants of RNNs are widely used to combat this issue.

LSTM Networks [97] use a mechanism called the Constant Error Carousel (CEC), which has an internal memory cell with a linear connection to itself across time-steps. This allows gradients to flow over long distances. The update equations for an LSTM depend on a number of intermediate states and are given by:

$$\begin{aligned} x_i &= W_{\mathcal{V}} w_i, \\ F_i &= \sigma(W_F[h_{i-1}; x_i] + b_F), \\ I_i &= \sigma(W_I[h_{i-1}; x_i] + b_I), \\ O_i &= \sigma(W_O[h_{i-1}; x_i] + b_O), \end{aligned} \qquad \begin{aligned} \tilde{c}_i &= \tanh(W_c[h_{i-1}; x_i] + b_c), \\ c_i &= F_i \odot c_{i-1} + I_i \odot \tilde{c}_i, \\ h_i &= O_i \odot \tanh(c_i). \end{aligned} \tag{2.2}$$

The first step above converts the one-hot word representation w_i to a *word embedding* x_i, by slicing a column from large matrix $W_{\mathcal{V}}$. Here σ is the sigmoid function and $\odot$ denotes element-wise multiplication. F, I, O are forget, input and output gates, respectively, and c_i is the internal memory state. $W_{\mathcal{V}} \in \mathbb{R}^{d \times |\mathcal{V}|}; W_F, W_I, W_O, W_c \in \mathbb{R}^{d \times 2d}; b_F, b_I, b_O, b_c \in \mathbb{R}^d$ are all learnable parameters.

GRU Networks [29] use a simpler mechanism to combat vanishing gradients. Starting from the word embeddings x_i:

$$\begin{aligned} r_i &= \sigma(W_r[h_{i-1}; x_i] + b_r), \\ z_i &= \sigma(W_z[h_{i-1}; x_i] + b_z), \\ \tilde{h}_i &= \tanh(W_h[r_i \odot h_{i-1}; x_i] + b_h), \\ h_i &= (1 - z_i) \odot h_{i-1} + z_i \odot \tilde{h}_i. \end{aligned} \tag{2.3}$$

Here, r_i is called the reset gate, and z_i the update gate. $W_r, W_z, W_h \in \mathbb{R}^{d \times 2d}; b_r, b_z, b_h \in \mathbb{R}^d$ are the learnable parameters.

In practice, both these variants are used almost interchangeably. Further, multiple layers of each may be stacked on top of each other where the output h_i from one layer forms the input for

the next layer. In the rest of this thesis, we will use $X = [x_1, \ldots, x_N]$ to denote the sequence of word embeddings; and $H = \overleftrightarrow{\text{LSTM}}(X)$ or $H = \overleftrightarrow{\text{GRU}}(X)$, where $H = [h_1, \ldots, h_N]$, to denote the full output of a (possibly multi-layer) bidirectional LSTM or GRU on top of it. In certain cases we will also use $h_N = \overrightarrow{\text{LSTM}}(X)$ or $h_N = \overrightarrow{\text{GRU}}(X)$ to denote the final output embedding (corresponding to x_N) from a unidirectional left-to-right LSTM or GRU.

Transformer Networks. The sequential nature of computation within an RNN prohibits parallelization across the length of the input. Also, despite improved gradient backpropagation in LSTMs and GRUs, modeling long-range dependencies still requires propagating signals for long distances. To overcome these limitations, Vaswani et al. [209] proposed the Transformer Network, which replaces recurrent connections with *self-attention*. The key idea is to model pairwise dependencies between all tokens in the input, followed by token-specific averages.

Starting again with a sequence of word embeddings X, we derive three different linear projections—queries Q, keys K and values V. The queries and keys are compared to derive attention scores between each pair of tokens, which are subsequently used to compute an averaged output representation of each token. The basic idea is as follows:

$$\begin{aligned} q_i = W_Q x_i, \quad k_i = W_K x_i, \quad v_i = W_V x_i, \qquad \forall i = 1, \ldots, N \\ \tilde{h}_i = W_O \sum_{j=1}^{N} \frac{\exp(q_i^T k_j)}{\sum_{j'} \exp(q_i^T k_{j'})} v_j \end{aligned} \tag{2.4}$$

Let $\tilde{H} = [\tilde{h}_1, \ldots, \tilde{h}_N]$ be a column-wise stacked version of the hidden states above. Then the output of the transformer layer is given by $H = \text{LayerNorm}(X + \tilde{H})$, where LayerNorm() is a layer normalization transformation [6]. In practice, $\tilde{H}$ is computed across multiple *heads*, by taking multiple projections of the input embeddings, and concatenating the outputs of self-attention. $W_Q, W_K, W_V, W_O \in \mathbb{R}^{d \times d}$ are all learnable parameters. Similar to RNNs, multiple transformer layers can be stacked on top of each other such that the output of one layer forms the input of the next one. The complexity of one layer is $O(N^2 d)$, compared to $O(N d^2)$ for an RNN, but importantly this can be parallelized across the N inputs.

Note that the summation in the self-attention mechanism does not depend on the positions of the different tokens in the input. In order to utilize the order of the sequence, the input word embeddings are summed with positional embeddings, which can be computed in multiple ways (see Vaswani et al. [209] for details). Transformer Networks have shown superior performance, both in terms of speed and accuracy, across a wide range of NLP tasks recently. While the methods presented in Chapters 3-6 use RNN variants, since they were developed before the

introduction of Transformers, in practice they can be easily replaced. In Chapters 7-8, we use Transformers.

2.2.2 Representing Graphs

Besides text, we will also be interested in deriving neural representations of graphical data [128, 173]. The standard approach for this involves using some variant of *Graph Convolution Networks (GCNs)* [110], which apply a convolution operation to local neighborhoods in the graph to derive node features. Many of these variants can be seen as special cases of the Message Passing Neural Network (MPNN) framework introduced by Gilmer et al. [78]. Given a directed or undirected graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, where $\mathcal{N}$ is the set of nodes, and $\mathcal{E} : \mathcal{N} \times \mathcal{N} \rightarrow \{0, 1\}$ is a set of edges, the goal is compute node representations $h_v \ \forall v \in \mathcal{N}$. The basic recipe for these models is as follows:

1. Initialize node representations $h_v^{(0)}$.
2. For $l = 1, \ldots, T$ update node representations

$$h_v^{(l)} = \phi\left(h_v^{(l-1)}, \sum_{v' \in N(v)} h_{v'}^{(l-1)}\right),$$

   where $N(v)$ denotes the neighbours of v, based on incoming edges, and ϕ is a parametric neural network layer.

Here T is the number of *layers* in the model and corresponds to the maximum length of the paths along which information should be propagated in the graph. Once the propagation is complete the final layer representations $h_v^{(T)}$ can be used to do tasks like node classification [110], or aggregated to an overall graph representation [64].

Relational GCNs. The KB formulation discussed above, is an instance of a *multi-relational* graph since its edges are typed, i.e. they map subject and object nodes to the relation between them $\mathcal{E} : \mathcal{N} \times \mathcal{N} \rightarrow \mathcal{R}$. In this case, we need to take into account the types of the edges when updating node representations [178]:

$$h_v^{(l)} = \phi\left(h_v^{(l-1)}, \sum_{R \in \mathcal{R}} \psi_R\left(\sum_{v' \in N_R(v)} h_{v'}^{(l-1)}\right)\right). \tag{2.5}$$

Here ψ_R is a relation-specific parametric layer, and $N_R(v)$ are nodes connected to v through relation R. Typically, for each relation we will also add its inverse to the graph, so that information can propagate from subjects to objects, as well as vice versa. In Chapter 5 we will

introduce an extension of relational-GCNs which can handle a heterogeneous graph consisting of both KB facts and text sentences.

2.2.3 Retrieval

The neural models discussed in the previous section all scale linearly with the input size. Hence, when trying to answer queries against a large knowledge source, we will often rely on shallow retrieval strategies to select a small relevant portion of it before running a more complex model. Here, we discuss general techniques for selecting subsets of text corpora and KBs based on a query.

TFIDF Search. Given a set of documents $\mathcal{S} = \{s_1, s_2, \ldots\}$, and a query q we can select the top-K relevant documents by defining a similarity function $F(q)^T F(s_i)$ between the query and each document. Here $F(x) \in \mathbb{R}^{\mathcal{V}}$ denotes a TFIDF vector constructed from the sequence of words in x as follows:

$$F(x)_i = \log(1 + tf(w_i, x)) \log(idf(w_i, \mathcal{S})) \quad \forall i = 1, \ldots, |\mathcal{V}|.$$

Here, $tf(w_i, x)$ is the *term-frequency*, or the count of w_i in x, and $idf(w_i, \mathcal{S})$ is the *inverse document frequency*, computed as [26]:

$$n_i = |\{s \in \mathcal{S} : w_i \in s\}|,$$
$$idf(w_i, \mathcal{S}) = \frac{|\mathcal{S}| - n_i + 0.5}{n_i + 0.5}.$$

In practice, for a given document or query the TFIDF vector $F(x)$ will mostly consist of zeros since the term frequency will be 0 for most words in $\mathcal{V}$. Hence, efficient algorithms for selecting the top-K relevant documents based on the above similarity function depend on either constructing an inverted index mapping words to the documents containing them,[4] or by using sparse matrix multiplication. Retrieval performance typically improves by expanding the vocabulary to include higher order n-grams, but this comes at a cost of increased computation.

Entity Linking. Given a text passage or a query s and the nodes in a KB $\mathcal{N}$, often we are interested in identifying a subset of nodes which are mentioned in s. If we are only interested in identifying the entities, but not the actual spans of text which refer to them, this can be framed as a text retrieval problem. The name of each entity in the KB can be considered as

[4] https://lucene.apache.org/

a document, and the top K entities which match a query can be retrieved using the TFIDF procedure described above. The number of entities retreived from $\mathcal{N}$ trades off precision with recall. If we are also interested in the exact span of text which refers to an entity, we can proceed in two steps. First, a named entity recognition (NER) system can be run to identify the entities in text. Next, the identified entities can each be linked separately to the nodes in the KB. Richer signals, based on hyperlinks, can be used when linking to Wikipedia, e.g. as done in the TagMe system [70].

Algorithm 1: Personalized Pagerank.

Input : Adjacency matrix $\hat{W}_{\mathcal{K}}$, starting distribution e_v.

Output: Pagerank scores p_v^*.

$r_0 = (1-\gamma)e_v$;

$p_v^* = r_0$;

for $t = 1, 2, \ldots$ *until convergence:* **do**

  $r_t = \gamma \hat{W}_{\mathcal{K}} r_{t-1}$;

  $p_v^* = p_v^* + r_t$;

end

Return p_v^*.

Personalized Pagerank [89]. Given the subset of nodes in a KB relevant to the query, denoted as $\mathcal{N}_q$, we can model a spreading activation process starting from these to identify a *subgraph* relevant to the query. Essentially, we compute a score called pagerank, which measures the similarity of any $v \in \mathcal{N}$ to the nodes in $\mathcal{N}_q$, and retain the ones with the maximum scores. In order to compute these scores, we represent our KB graph using an adjacency matrix $W_{\mathcal{K}}$ of size $|\mathcal{N}| \times |\mathcal{N}|$, s.t. $W_{\mathcal{K}}[i,j] = 1$ iff there is an edge from node j to node i. Let $\hat{W}_{\mathcal{K}}$ be a column-normalized version of this matrix. Then the pagerank scores are given by a distribution $p_v^* \in \mathbb{R}^{|\mathcal{N}|}$, s.t.

$$p_v^* = (1-\gamma)e_v + \gamma \hat{W}_{\mathcal{K}} p_v^*.$$

Here, $e_v \in \mathbb{R}^{|\mathcal{N}|}$ is a uniform starting distribution over the nodes in $\mathcal{N}_q$, and γ is a hyperparameter controlling how the weights spread starting from the initial set of nodes. A simple procedure for computing p_v^*, based on power iteration, is outlined in Algorithm 1 [34]. The basic idea consists of uniformly distributing some mass on the nodes in $\mathcal{N}_q$, and at each step

distributing the mass of a node to its neighbors. The algorithm is guaranteed to converge. We can also use a weighted version of $\hat{W}_{\mathcal{K}}$ which determines the relative importance of edges.

Maximum Inner Product Search. In Chapters 7 & 8 we will consider methods which directly retrieve answers by comparing a dense embedding of the query $e_q \in \mathbb{R}^d$ to a large set of embeddings $E \subset \mathbb{R}^d$. An exact search will scale linearly with the number of points in E, but approximate search can be done in sub-linear time. The basic idea behind these techniques comes from *Locality Sensitive Hashing* [79], where the points in E are mapped to buckets using hash functions such that nearby points, based on some distance metric, end up in the same bucket. This involves computing an offline index from all the embeddings in E. At inference time, the query is also mapped to one such bucket, and an exact search is then done over the points only within that bucket. We refer readers to Shrivastava and Li [187] and Andoni et al. [1] for a detailed discussion of the algorithms.

Part I

Learning to Read

Chapter 3

Reading Comprehension

A fundamental challenge in using text as a source of knowledge lies in understanding natural language. In this chapter we will focus on *machine reading*, the autonomous understanding of text [68], from the perspective of answering questions posed over a text passage. Standardized QA tests are useful benchmarks for measuring both machine and human reading comprehension [33], and in this chapter we will leverage large-scale training data to build systems for this task with minimal manual engineering. For now, we will assume that given a question, we know the passage which contains its answer. Later, in Chapter 5 we will relax this assumption. The work described in this chapter first appeared in the following two publications:

- Bhuwan Dhingra, Hanxiao Liu, Zhilin Yang, William Cohen, and Ruslan Salakhutdinov. Gated-attention readers for text comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1832–1846, 2017
- Bhuwan Dhingra, Qiao Jin, Zhilin Yang, William Cohen, and Ruslan Salakhutdinov. Neural models for reasoning over multiple mentions using coreference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 42–48, 2018

Code for reproducing the experiments in this chapter is available on GitHub.[1,2]

[1] https://github.com/bdhingra/ga-reader
[2] https://github.com/bdhingra/coref-gru

3.1 Overview

Specifically, the task we are interested in involves tuples of the form $(s, q, a, \mathcal{C})$, where the goal is to find the answer a to the question q from the candidates $\mathcal{C}$ with the document s as context. Both the question q and document s are sequences of words (tokens). The candidate answers $\mathcal{C}$ may be extracted from a separate pipeline and known before-hand, or they may consist of all spans in s up to a maximum length. Either way, we will assume each candidate $c \in \mathcal{C}$ has at least one token which also appears in s. In the last few years, several large-scale reading comprehension datasets have been released which fit this definition [94, 95, 104, 151, 166].

Deep learning models which leverage this data outperform traditional shallow approaches on this task [94]. These models learn contextual representations of the tokens in the document, which are matched with a representation of the question to compute the probability of each token whether it answers the question. Two factors which have been shown to be important in this design are: (1) *Multi-hop architectures* [183, 190, 228], which allow a model to scan the document and the question iteratively in multiple passes. (2) *Attention mechanisms* [25, 94], borrowed from the machine translation literature [7], which allow the model to focus on appropriate subparts of the document based on the query. Intuitively, the multi-hop architecture allows the reader to incrementally refine token representations, and the attention mechanism re-weights different parts in the document according to their relevance to the query.

In § 3.2 we introduce a neural architecture which combines these two aspects in a complementary manner. Our main contribution is a novel attention mechanism, called Gated-Attention (GA), which gates the evolving representations of tokens in the document across hops. More specifically, unlike the conventional use of attention which *aggregates* either token [25, 94, 95, 106] or sentence [194, 228] representations, the GA module instead *updates* the representations based on the query, using an element-wise multiplication operation. Since this keeps the sizes of the representations unchanged, it can be applied in between the multiple layers of the model, which enables learning conditional token representations w.r.t. the question, and leads to more accurate answer selection (§ 3.4).

The layers themselves are based on RNNs introduced in the last chapter to compute the document token representations. RNN layers have a bias towards *sequential-recency* [65], i.e. a tendency to favor short-term dependencies. This leads to poor performance on problems which require reasoning about information in different parts of the text. One important form of reasoning for Question Answering (QA) models is the ability to aggregate information from multiple mentions of entities, which we term as *coreference-based reasoning*. Coreference is the

Context: [...] **mary** got the football there [...] **mary** went to the bedroom [...] **mary** travelled to the hallway [...]
Question: where was the football before the hallway ?

Context: Louis-Philippe Fiset [...] was a local physician and politician in the **Mauricie** area [...] is located in the **Mauricie** region of Quebec, Canada [...]
Question: country of citizenship – louis-philippe fiset ?

Figure 3.1: Example questions which require coreference-based reasoning from the bAbi dataset [227] (top) and Wikihop dataset [223] (bottom). Coreferences are in bold, and the correct answers are underlined.

phenomenon where different expressions in the text refer to the same underlying real-world entity. Figure 3.1 shows examples.

Attention mechanisms like GA alleviate part of the issue, but empirical studies suggest RNNs with attention also have difficulty modeling long-term dependencies [40]. This problem becomes even more severe when training data is scarce, and inductive biases play an important role. Hence, in § 3.3, we explore using an external coreference resolution system to model coreferent dependencies in an RNN. Coreference resolution has seen a gradual increase in accuracy over the years [62, 119, 235], and we propose to use coreference annotations to adapt the standard RNN layer by introducing a bias towards *coreferent-recency*. Specifically, we introduce a term in the update equations for GRUs which depends on the hidden state of the coreferent antecedent of the current token (if it exists). This way hidden states are propagated along coreference chains and the original sequence in parallel.

3.2 Gated-Attention Reader

The contextual representations in GA reader, namely the embeddings of words in the document, are iteratively refined across hops using bidirectional GRUs until reaching a final attention-sum module [106] which maps the contextual representations in the last hop to a probability distribution over candidate answers. In reading comprehension tasks, ideally, the semantic meaning carried by the contextual embeddings should be aware of the query across hops. As an example, human readers are able to keep the question in mind during multiple passes of reading, to successively mask away information irrelevant to the query. Hence, we propose a fine-grained attention model, called *gated-attention*, which attends to components of the semantic represen-

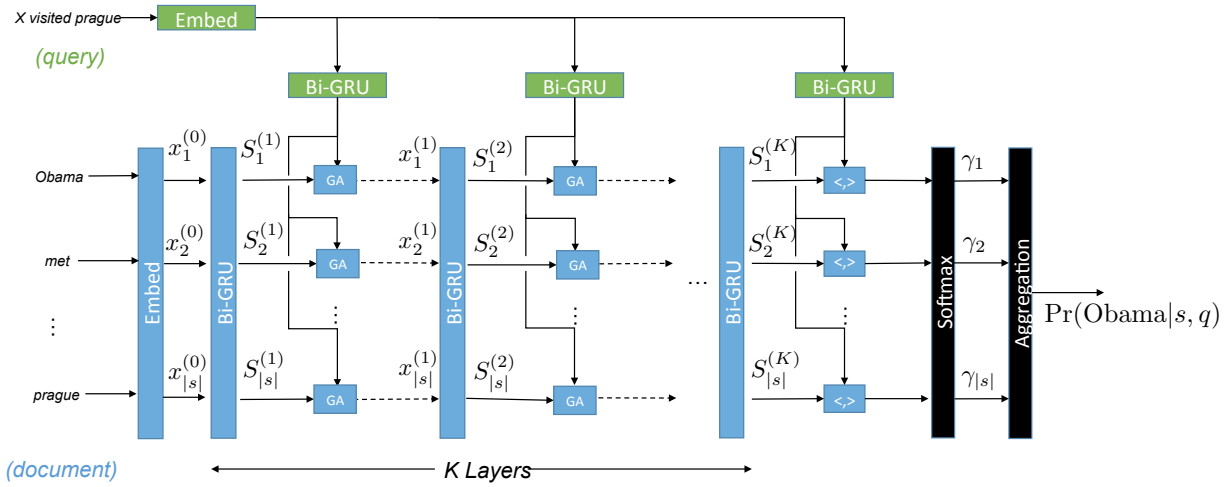


Figure 3.2: Gated-Attention Reader. Dashed lines represent dropout connections.

tation of the document being built up by the GRU. Gated-attention is implemented via *multiplicative* interactions between the query and the contextual embeddings, and is applied in each layer as a *filtering* process. The filters weigh individual components of the vector representation of *each* token in the document separately.

The design of gated-attention layers is motivated by the effectiveness of multiplicative interaction among vector-space representations, e.g., in various types of recurrent units [97, 238] and in relational learning [111, 242]. While other types of compositional operators are possible, such as concatenation or addition [141], we found that multiplication has strong empirical performance.

Preliminaries

Let $X^{(0)} = [x_1^{(0)}, x_2^{(0)}, \ldots x_{|s|}^{(0)}]$ denote the token embeddings of the document, which are also inputs at layer 1 for the document reader below, and $Y = [y_1, y_2, \ldots y_{|q|}]$ denote the token embeddings of the query. These embeddings can come from a pre-trained word embedding table, or initialized randomly. Here $|s|$ and $|q|$ denote the document and query lengths respectively.

Multi-Hop Architecture

Fig. 3.2 illustrates the Gated-Attention (GA) reader. The model reads the document and the query over K layers, where layer k receives the contextual embeddings $X^{(k-1)}$ of the document from the previous layer. The document embeddings are transformed by taking the full output

of a document Bi-GRU (indicated in blue in Fig. 3.2):

$$S^{(k)} = \overleftrightarrow{\text{GRU}}_s^{(k)}(X^{(k-1)}). \tag{3.1}$$

At the same time, a layer-specific query representation is computed as the full output of a separate query Bi-GRU (indicated in green in Figure 3.2):

$$Q^{(k)} = \overleftrightarrow{\text{GRU}}_q^{(k)}(Y). \tag{3.2}$$

Next, *Gated-Attention* is applied to $S^{(k)}$ and $Q^{(k)}$ to compute inputs for the next layer $X^{(k)}$.

$$X^{(k)} = \text{GA}(S^{(k)}, Q^{(k)}),$$

where GA is defined in the following subsection.

Gated-Attention Module

For brevity, let us drop the superscript k in this subsection as we are focusing on a particular layer. Let S_i be the representation of the i-th token in s at this layer. The GA module forms a token-specific representation of the query $\tilde{q}_i$ using soft attention, and then multiplies the query representation element-wise with the document token representation. Specifically, for $i = 1, \ldots, |s|$:

$$\alpha_i = \text{softmax}(Q^\top S_i), \tag{3.3}$$

$$\tilde{q}_i = Q\alpha_i,$$

$$x_i = S_i \odot \tilde{q}_i. \tag{3.4}$$

Here, softmax is a normalization function defined over a sequence as follows:

$$\text{softmax}(a_1, a_2, \ldots)_i = \frac{\exp(a_i)}{\sum_{i'} \exp(a_{i'})}.$$

$X = [x_1, \ldots, x_{|s|}]$ forms the input for the next layer's bidirectional GRU.

Answer Prediction

We consider both natural questions, which typically start with a wh-word, and *cloze* questions, which are regular sentences but with a missing token which needs to be filled out. For the former, let $q^{(K)} = [q^f_{|q|}; q^b_0]$ be the output of the final layer query Bi-GRU, obtained by concatenating

the final states of the forward and backward GRUs. For the latter, let $q^{(K)} = [q^f_\ell; q^b_{T-\ell+1}]$ be an intermediate output of the final layer query Bi-GRU at the location ℓ of the cloze token in the query. Also, let $S^{(K)} = \overleftrightarrow{\text{GRU}}^{(K)}_s(X^{(K-1)})$ be the full output of final layer document Bi-GRU.

To obtain the probability that a particular token in the document answers the query, we take an inner-product between these two, and pass it through a softmax layer:

$$\gamma = \text{softmax}((S^{(K)})^T q^{(K)})$$

where vector $\gamma \in \mathbb{R}^{|s|}$ defines a probability distribution over the $|s|$ tokens in the document. The probability of a particular candidate $c \in \mathcal{C}$ as being the answer is then computed by aggregating the probabilities of all document tokens which appear in c and renormalizing over the candidates:

$$\Pr(c|s,q) \propto \sum_{i \in \mathbb{I}(c,s)} \gamma_i \tag{3.5}$$

where $\mathbb{I}(c,s)$ is the set of positions where a token in c appears in the document s. This aggregation operation is the same as the *pointer sum attention* applied in the AS Reader [106]. Finally, the candidate with maximum probability is selected as the answer:

$$a^* = \text{argmax}_{c \in \mathcal{C}} \ \Pr(c|s,q).$$

During the training phase, given a dataset $\mathcal{D}$, model parameters are updated w.r.t. a cross-entropy loss between the predicted probabilities and the true answers:

$$\mathcal{L} = \sum_{(s,q,a,\mathcal{C}) \in \mathcal{D}} -\log \Pr(a|s,q).$$

Further Enhancements

Character-level Embeddings: Given a token w from the document or query, its vector space representation can be computed as $x = [L(w); C(w)]$. $L(w)$ retrieves the word-embedding for w from a lookup table $L \in \mathbb{R}^{d \times \mathcal{V}}$, whose columns hold a vector for each unique token in the vocabulary. We can also utilize a character composition model $C(w)$ which generates an orthographic embedding of the token. Such embeddings have been shown to be helpful for tasks like Named Entity Recognition [243] and dealing with OOV tokens at test time [52]. The embedding $C(w)$ is generated by taking the final output of a Bi-GRU applied to character embeddings in the token and applying a linear transformation:

$$z = \overleftrightarrow{\text{GRU}}_c(w),$$
$$C(w) = Wz + b.$$

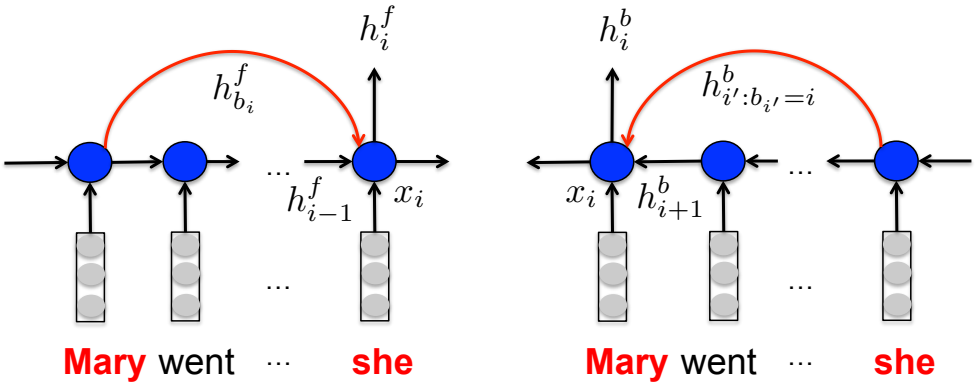


Figure 3.3: Forward (left) and Backward (right) Coref-RNN layers. *Mary* and *she* are coreferent.

Question Evidence Common Word Feature (qe-feature): Li et al. [124] proposed a simple token level indicator feature which significantly boosts reading comprehension performance in some cases. For each token in the document we construct a one-hot vector $f_i \in \{0,1\}^2$ indicating its presence in the query. This vector can be incorporated into the GA reader by assigning a feature lookup table $F \in \mathbb{R}^{n_F \times 2}$ (we use $n_F = 2$), taking the feature embedding $e_i = F f_i$ and appending it to the inputs of the last layer document BiGRU as, $[x_i^{(K)}; e_i]$ for all i. We conduct our experiments both with and without this feature.

3.3 Extending with Coreference

Next, we discuss how to extend the above model with coreference information to improve its modeling of long-term dependencies. We will focus on the recurrent layer (Eq. 2.3), and keep the rest of the model unchanged. Specifically, we introduce the concept for *coreferent-recency*, in contrast to *sequential-recency* inherent in RNNs.

Coref-RNNs. Suppose we are given an input sequence $s_1, s_2, \ldots$ along with their word vectors $x_1, x_2, \ldots$ and annotations for the most recent coreferent antecedent for each token $b_1, b_2, \ldots$, where $b_i \in \{0, \ldots, i-1\}$ and $b_i = 0$ denotes the null antecedent (for tokens not belonging to any cluster). Further, we assume all tokens belonging to a mention in a cluster belong to that cluster, and there are B clusters in total. We can use an off-the-shelf coreference tool, such as Stanford CoreNLP,[3] to obtain these annotations.

[3] `https://stanfordnlp.github.io/CoreNLP/`

The update rule in any RNN takes the same basic form given by (from Eq. 2.1):

$$h_i = f(Wx_i + Uh_{i-1} + c).$$

The bias for sequential-recency comes from the second term Uh_{i-1}. In this work we add another term to introduce a bias towards coreferent-recency instead:

$$h_i = f(Wx_i + \alpha_i U\phi_s(h_{i-1}) + (1-\alpha_i)U'\phi_c(h_{b_i}) + c),$$

where h_{b_i} is the hidden state of the coreferent antecedent of s_i (with $h_0 = 0$), ϕ_s and ϕ_c are non-linear functions applied to the hidden states coming from the sequential antecedent and the coreferent antecedent, respectively, and α_i is a scalar weight which decides the relative importance of the two terms based on the current input (so that, for example, pronouns may assign a higher weight for the coreferent state). When $b_i = 0$, α_i is set to 1, otherwise it is computed using a key-based addressing scheme [135], as $\alpha_i = \text{softmax}(x_i^T k)$, where k is a trainable key vector. In this work we use simple slicing functions $\phi_s(x) = x[1 : d/2]$, and $\phi_c(x) = x[d/2 : d]$ which decompose the hidden states into a sequential and a coreferent component, respectively. Figure 3.3 (left) shows an illustration of the layer.

Coref-GRU (C-GRU). The above extension to RNNs can be applied to any recurrent layer; here we will focus on GRU cells (Eq. 2.3). For simplicity, we introduce the variable m_i which concatenates the sequential and coreferent hidden states:

$$m_i = [\alpha_i\phi_s(h_{i-1}); (1-\alpha_i)\phi_c(h_{b_i})].$$

Then the update equations are given by:

$$\begin{aligned}
r_i &= \sigma(W^r x_i + U^r m_i + c^r),\\
z_i &= \sigma(W^z x_i + U^z m_i + c^z),\\
\tilde{h}_i &= \tanh(W^h x_i + r_i \odot U^h m_i + c^h),\\
h_i &= (1-z_i)\odot m_i + z_i \odot \tilde{h}_i.
\end{aligned}$$

The attention parameter α_i is given by:

$$\alpha_i = \frac{\exp x_i^i k_1}{\exp x_i^i k_1 + \exp x_i^i k_2},$$

where k_1 and k_2 are trainable key vectors.

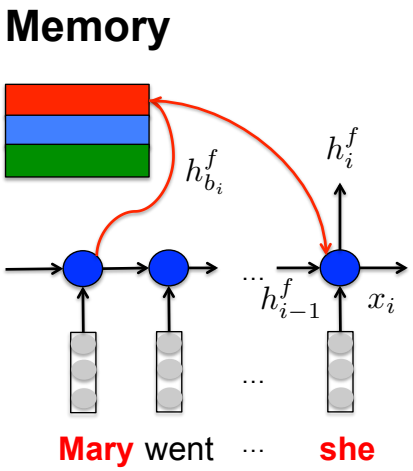


Figure 3.4: Alternative view of Coref-RNNs as a memory network. Each memory cell corresponds to a coreference cluster, and read / write operations are performed on it when a token from that cluster is encountered in the text.

Connection to Memory Networks. Coref-RNNs can be viewed as special cases of memory networks [194] with a memory state M_i at each time step which is a $B \times d$ matrix (Figure 3.4). The rows of this memory matrix correspond to the state of each coreference cluster at time step i. The main difference between Coref-RNNs and a typical memory network such as EntNets [91] is that we use coreference annotations to read and write from the memory rather than let the model learn how to access the memory. With Coref-RNNs, only the content of the memories needs to be learned. As we shall see in § 3.4, this turns out to be a useful bias in a low-data regime.

Bidirectional C-GRU. To extend to the bidirectional case, a second layer is fed the same sequence in the reverse direction (Figure 3.3). Now the coreferent state at time-step i comes from the immediate descendent of a token, i.e. $h_{i'}$, where $b_{i'} = i$ and $i' > i$. Outputs from the two layers are then concatenated in a similar manner to bidirectional LSTMs and GRUs.

Complexity. The resulting layer has the same time-complexity as that of a regular RNN layer. The memory complexity increases since we have to keep track of the hidden states for each coreference cluster in the input. If there are B clusters and N is the batch size, the resulting complexity is by $O(N|S|Bd)$.

	CNN	Daily Mail	CBT-NE	CBT-CN	WDW-Strict	WDW-Relaxed
# train	380,298	879,450	108,719	120,769	127,786	185,978
# validation	3,924	64,835	2,000	2,000	10,000	10,000
# test	3,198	53,182	2,500	2,500	10,000	10,000
# vocab	118,497	208,045	53,063	53,185	308,602	347,406
max doc length	2,000	2,000	1,338	1,338	3,085	3,085

Table 3.1: Cloze-style QA dataset statistics.

3.4 Experiments

3.4.1 Cloze-style QA

We start by experimenting with synthetic queries, where a token or an entity is masked in a sentence and must be predicted from a separate context. Such queries can often be constructed automatically using smart heuristics and minimal effort, and are also easy to evaluate in terms of accuracy of predicting the masked token. For these datasets we use regular RNNs; experiments evaluating Coref-RNNs are presented in the next three subsections.

Datasets

We evaluate the GA reader on five large-scale datasets. The first two, CNN and Daily Mail news stories consist of articles from the popular CNN and Daily Mail websites [94].[4] A query over each article is formed by removing an entity from the short summary which follows the article. Further, entities within each article were anonymized to make the task purely a comprehension one. N-gram statistics, for instance, computed over the entire corpus are no longer useful in such an anonymized corpus.

The next two datasets are formed from two different subsets of the Children's Book Test (CBT) [95].[5] Documents consist of 20 contiguous sentences from the body of a popular children's book, and queries are formed by deleting a token from the 21st sentence. We only focus on subsets where the deleted token is either a common noun (CBT-CN) or named entity (CBT-NE) since simple language models already give human-level performance on the other types [95].

[4] https://github.com/deepmind/rc-data
[5] https://research.fb.com/downloads/babi/

Hyperparameter	CNN	Daily Mail	CBT-NE	CBT-CN	WDW-Strict	WDW-Relaxed
Dropout	0.2	0.1	0.4	0.4	0.3	0.3
$\dim(\overleftrightarrow{\text{GRU}}_*)$	256	256	128	128	128	128

Table 3.2: Hyperparameter settings for each dataset. dim() indicates hidden state size of GRU.

The final dataset is Who Did What (WDW) [151], constructed from the LDC English Gigaword newswire corpus.[6] First, article pairs which appeared around the same time and with overlapping entities are chosen, and then one article forms the document and a cloze query is constructed from the other. Missing tokens are always person named entities. Questions which are easily answered by simple baselines are filtered out, to make the task more challenging. There are two versions of the training set—a small but focused "Strict" version and a large but noisy "Relaxed" version. We report results on both settings which share the same validation and test sets.

Implementation Details

The model was implemented using the Theano [204] and Lasagne Python libraries.[7] We used stochastic gradient descent with ADAM updates for optimization, which combines classical momentum and adaptive gradients [109]. The batch size was 32 and the initial learning rate was 5×10^{-4} which was halved every epoch after the second epoch. The same setting is applied to all models and datasets. We also used gradient clipping with a threshold of 10 to stabilize GRU training [153]. We set the number of layers K to be 3 for all experiments. The number of hidden units for the character GRU was set to 50. The remaining two hyperparameters—size of document and query GRUs, and dropout rate—were tuned on the validation set, and their optimal values are shown in Table 3.2. In general, the optimal GRU size increases and the dropout rate decreases as the corpus size increases.

The word lookup table was initialized with $100d$ GloVe vectors [155] and OOV tokens at test time were assigned unique random vectors (we discuss these choices in more detail in the next chapter).[8] We empirically observed that initializing with pre-trained embeddings gives higher performance compared to random initialization for all datasets. Furthermore, for smaller datasets (WDW and CBT) we found that fixing these embeddings to their pretrained values led

[6] https://tticnlp.github.io/whodidwhat/
[7] https://lasagne.readthedocs.io/en/latest/
[8] http://nlp.stanford.edu/projects/glove/

to higher test performance, possibly since it avoids overfitting. We do not use the character composition model for CNN and Daily Mail, since their entities (and hence candidate answers) are anonymized to generic tokens. For other datasets the character lookup table was randomly initialized with $25d$ vectors. All other parameters were initialized to their default values as specified in the Lasagne library.

Performance Comparison

Tables 3.3 and 3.4 show a comparison of the performance of GA Reader with previously published models. The numbers reported for GA Reader are for single models, though we compare to both ensembles and single models from prior work. We present 4 variants of the GA Reader, using combinations of whether the qe-feature is used or not, and whether the word lookup table $L(w)$ is updated during training or fixed to its initial value.

Interestingly, we observe that feature engineering leads to significant improvements for WDW and CBT datasets, but not for CNN and Daily Mail datasets. We note that anonymization of the latter datasets means that there is already some feature engineering (it adds hints about whether a token is an entity), and these are much larger than the other four. In machine learning it is common to see the effect of feature engineering diminish with increasing data size. Similarly, fixing the word embeddings provides an improvement for WDW and CBT, but not for CNN and Daily Mail. This is not surprising given that the latter datasets are larger and less prone to overfitting.

Comparing with prior work, on the WDW dataset the basic version of the GA Reader outperforms all previously published models when trained on the Strict setting. By adding the qe-feature the performance increases by 3.2% and 3.5% on the Strict and Relaxed settings respectively. On the CNN and Daily Mail datasets the GA Reader leads to an improvement of 3.2% and 4.3% respectively over the best previous single models, as well as ensemble models. For CBT-NE, GA Reader with the qe-feature outperforms all previous single and ensemble models except the AS Reader trained on the much larger BookTest Corpus [8]. Lastly, on CBT-CN the GA Reader with the qe-feature outperforms all previously published single models except the NSE, and AS Reader trained on a larger corpus. For each of the 4 datasets on which GA achieves the top performance, we conducted one-sample proportion tests to test whether GA is significantly better than the second-best baseline. The p-values are 0.319 for CNN, <0.00001 for DailyMail, 0.028 for CBT-NE, and <0.00001 for WDW. Hence, GA significantly outperforms all other baselines on 3 out of those 4 datasets at the 5% level.

Model	CNN		Daily Mail		CBT-NE		CBT-CN	
	Val	Test	Val	Test	Val	Test	Val	Test
Humans (query) †	–	–	–	–	–	52.0	–	64.4
Humans (context + query) †	–	–	–	–	–	81.6	–	81.6
LSTMs (context + query) [95] †	–	–	–	–	51.2	41.8	62.6	56.0
Deep LSTM Reader [94] †	55.0	57.0	63.3	62.2	–	–	–	–
Attentive Reader [94] †	61.6	63.0	70.5	69.0	–	–	–	–
Impatient Reader [94] †	61.8	63.8	69.0	68.0	–	–	–	–
MemNets [228] †	63.4	66.8	–	–	70.4	66.6	64.2	63.0
AS Reader [106] †	68.6	69.5	75.0	73.9	73.8	68.6	68.8	63.4
DER Network [112] †	71.3	72.9	–	–	–	–	–	–
Stanford AR (relabeling) [25] †	73.8	73.6	77.6	76.6	–	–	–	–
Iterative Attentive Reader [190] †	72.6	73.3	–	–	75.2	68.6	72.1	69.2
EpiReader [207] †	73.4	74.0	–	–	75.3	69.7	71.5	67.4
AoA Reader [39] †	73.1	74.4	–	–	77.8	72.0	72.2	69.4
ReasoNet [184] †	72.9	74.7	77.6	76.6	–	–	–	–
NSE [145] †	–	–	–	–	78.2	73.2	74.3	**71.9**
BiDAF [179] †	76.3	76.9	80.3	79.6	–	–	–	–
MemNets (ensemble) [228] †	66.2	69.4	–	–	–	–	–	–
AS Reader (ensemble) [106] †	73.9	75.4	78.7	77.7	76.2	71.0	71.1	68.9
Stanford AR (relabeling,ensemble) [25] †	77.2	77.6	80.2	79.2	–	–	–	–
Iterative Attentive Reader (ensemble) [190] †	75.2	76.1	–	–	76.9	72.0	74.1	71.0
EpiReader (ensemble) [207] †	–	–	–	–	76.6	71.8	73.6	70.6
AS Reader (+BookTest) [8] † ‡	–	–	–	–	80.5	76.2	83.2	80.8
AS Reader (+BookTest,ensemble) [8] † ‡	–	–	–	–	*82.3*	*78.4*	*85.7*	*83.7*
GA (update $L(w)$)	**77.9**	**77.9**	**81.5**	**80.9**	76.7	70.1	69.8	67.3
GA (fix $L(w)$)	77.9	77.8	80.4	79.6	77.2	71.4	71.6	68.0
GA (+qe-feature, update $L(w)$)	77.3	76.9	80.7	80.0	77.2	73.3	73.0	69.8
GA (+qe-feature, fix $L(w)$)	76.7	77.4	80.0	79.3	**78.5**	**74.9**	**74.4**	70.7

Table 3.3: Validation/Test accuracy $(\%)$ on CNN, Daily Mail and CBT. Results marked with "†" are cf previously published works. Results marked with "‡" were obtained by training on a larger training set. Best performance on standard training sets is in bold, and on larger training sets in italics.

Model	Strict		Relaxed	
	Val	Test	Val	Test
Human [151] †	–	84	–	–
Attentive Reader [151] †	–	53	–	55
AS Reader [151] †	–	57	–	59
Stanford AR [151] †	–	64	–	65
NSE [145] †	66.5	66.2	67.0	66.7
GA (update $L(w)$)	67.8	67.0	67.0	66.6
GA (fix $L(w)$)	68.3	68.0	69.6	69.1
GA (+qe-feature, update $L(w)$)	70.1	69.5	70.9	71.0
GA (+qe-feature, fix $L(w)$)	**71.6**	**71.2**	**72.6**	**72.6**

Table 3.4: Validation/Test accuracy (%) on WDW dataset for both "Strict" and "Relaxed" settings. Results with "†" are cf previously published works.

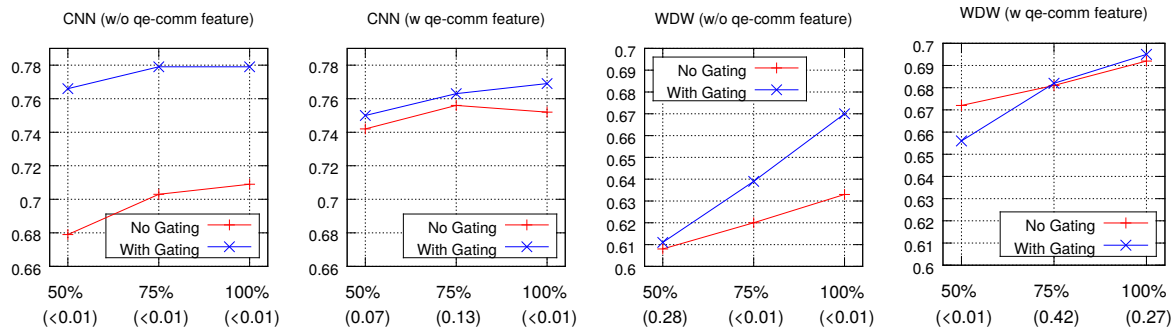


Figure 3.5: Performance in accuracy with and without the Gated-Attention module over different training sizes. p-values for an exact one-sided Mcnemar's test are given inside the parentheses for each setting.

Ablations

In this section we do an ablation study to see the effect of Gated Attention. We compare the GA Reader as described here to a model which is exactly the same in all aspects, except that it passes document embeddings $D^{(k)}$ in each layer directly to the inputs of the next layer without using the GA module. In other words $X^{(k)} = D^{(k)}$ for all $k > 0$. This model ends up using only one query GRU at the output layer for selecting the answer from the document. We compare

Gating	Accuracy	
	Val	Test
Sum	64.9	64.5
Concatenate	64.4	63.7
Multiply	**68.3**	**68.0**

K	Accuracy	
	Val	Test
1	–	57
2	65.6	65.6
3	**68.3**	68.0
4	**68.3**	**68.2**

Model	Accuracy	
	Val	Test
GA	**68.3**	**68.0**
−char	66.9	66.9
−qry-attn	65.7	65.0
+corpus-emb	64.0	62.5

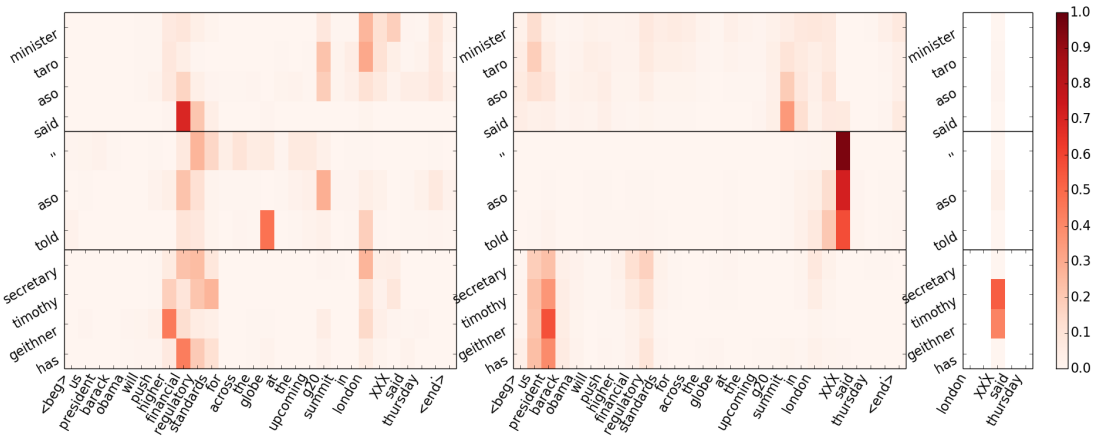
Table 3.5: Performance of several model variants on WDW dataset without using qe-comm feature and with fixed $L(w)$. (Left) Gating functions in the GA module. (Middle) Number of hops K. (Right) Model variants.

these two variants both with and without the qe-feature on CNN and WDW datasets for three subsets of the training data - 50%, 75% and 100%. Test set accuracies for these settings are shown in Figure 3.5. On CNN when tested without feature engineering, we observe that GA provides a significant boost in performance compared to without GA. When tested with the feature it still gives an improvement, but the improvement is significant only with 100% training data. On WDW-Strict, which is a third of the size of CNN, without the qe-feature we see an improvement when using GA versus without using GA, which becomes significant as the training set size increases. When tested with the feature on WDW, for a small data size without GA does better than with GA, but as the dataset size increases they become equivalent. We conclude that GA provides a boost in the absence of feature engineering, or as the training set size increases.

Next we look at the question of how to gate intermediate document reader states from the query, i.e. what operation to use in Eq. 3.4. Table 3.5 (Left) shows the performance on WDW dataset for three common choices: sum ($x = s + q$), concatenate ($x = s \| q$) and multiply ($x = s \odot q$). Empirically we find element-wise multiplication does significantly better than the other two.

In Table 3.5 (Middle) we show the effect of varying the number of hops K of the GA Reader on the final performance. We note that for $K = 1$, our model is equivalent to the AS Reader without any GA modules. We see a steep and steady rise in accuracy as the number of hops is increased from $K = 1$ to 3, which remains constant beyond that. This is a common trend in machine learning as model complexity is increased, however we note that a multi-hop architecture is important to achieve high performance on this task.

Table 3.5 (Right) shows accuracy on WDW by removing one component at a time. The



DOC: japanese prime minister taro aso said friday he would call for stronger monitoring of international finance at the g20 summit next week in london . `` we will h ave to emphatically argue that the foundation of the international monetary fund (imf) is weak and that we must establish financial regulations and supervisio n , '' aso told a legislative session . other world leaders have also pushed for stricter regulations of risky and unrestrained investment practices and instrum ents blamed for triggering the current global economic crisis . japan officially agreed in february to lend up to 100 billion dollars to the imf to provide fina ncial lifelines to emerging economies hit hard by the worldwide downturn . us treasury secretary timothy geithner has said president barack obama would discuss new global financial regulatory standards at the london summit .

QRY: <beg> us president barack obama will push higher financial regulatory standards for across the globe at the upcoming g20 summit in london , XXX said thursday . <end>
ANS: timothy geithner

Figure 3.6: Layer-wise attention visualization of GA Reader trained on WDW-Strict.

steepest reduction is observed when we replace pretrained GloVe vectors with those pretrained on the corpus itself ("+corpus-emb"). GloVe vectors were trained on a large corpus of about 6 billion tokens [155], and provide an important source of prior knowledge for the model. Note that the strongest baseline on WDW, NSE [146], also uses pretrained GloVe vectors, hence the comparison is fair in that respect. Next, we observe a substantial drop when removing token-specific attentions over the query in the GA module ("−qry-attn"), which allow gating individual tokens in the document only by parts of the query relevant to that token rather than the overall query representation. Finally, removing the character embeddings, which were only used for WDW and CBT, leads to a reduction of about 1% in the performance ("−char").

Analysis

To gain an insight into the reading process employed by the model we analyzed the attention distributions at intermediate layers of the reader. Figure 3.6 shows an example from the validation set of WDW dataset. In each figure, the left and middle plots visualize attention over the query (equation 3.3) for candidates in the document after layers 1 & 2 respectively. The right plot shows attention over candidates in the document of the cloze placeholder in the query (XXX) at the final layer. The full document, query and correct answer are shown at the bottom.

A generic pattern observed in such examples is that in intermediate layers, candidates in the document (shown along rows) tend to pick out salient tokens in the query which provide clues

about the cloze token, and in the final layer the candidate with the highest match with these tokens is selected as the answer. In Figure 3.6 there is a high attention of the correct answer on *financial regulatory standards* in the first layer, and on *us president* in the second layer. The incorrect answer, in contrast, only attends to one of these aspects, and hence receives a lower score in the final layer despite the n-gram overlap it has with the cloze token.

3.4.2 Reasoning Tasks

In this section we focus on three datasets (bAbi, Wikihop, LAMBADA) which explicitly require reasoning over long-term dependencies in the text, and experiment with using either GRU or Coref-GRU layers in the GA Reader for these tasks.

BAbi

Method	Avg	Max	# failed
EntNets [91]	–	0.704	15
QRN [180]	–	**0.901**	7
Bi-GRU	0.727	0.767	13
Bi-C-GRU	0.790	0.831	12
GA w/ GRU	0.764	0.810	10
GA w/ GRU + 1-hot	0.766	0.808	9
GA w/ C-GRU	0.870	0.886	**5**

Table 3.6: Accuracy on bAbi-1K, averaged across all 20 tasks. Following previous work we run each task for 10 random seeds, and report the Avg and Max (based on the dev set). A task is considered failed if its Max performance is < 0.95.

Dataset. Our first set of experiments are on the 1K training version of the synthetic bAbi AI tasks [227]. The passages and questions in this dataset are generated using templates, removing many complexities inherent in natural language, but it still provides a useful testbed for us since some tasks are specifically constructed to test the coreference-based reasoning we tackle here.

Implementation. We use a hidden state size of 64, batch size of 32, and learning rate 0.01 which is halved after every 120 updates. We also use dropout with rate 0.1 at the output of

each layer. The maximum number of coreference clusters across all tasks was $B = 13$. Since the passages are generated synthetically, we use a predefined dictionary of named entities to extract the coreference clusters. Half of the tasks in this dataset are extractive, meaning the answer is present in the passage, whereas the other half are classification tasks, where the answer is in a list of candidates which may not be in the passage. For the extractive tasks, we use the attention sum layer as described in the previous section. For the classification tasks we replace this with a softmax layer for predicting one of the classes. We extracted coreference clusters using an exact string match against a predefined set of entities.

Results. Table 3.6 shows a comparison of EntNets [91], Query Reduction Networks (QRNs) [180] and our models. QRNs employ a recurrent network over sentences in the text which jointly updates the query and sentence representations. We also include the results for a single layer version of GA Reader to enable fair comparison with EntNets. We denote this model simply as Bi-GRU, or Bi-C-GRU if using Coref-GRUs, since there is no Gated-Attention mechanism in this case. In each case we see clear improvements of using C-GRU layers over GRU layers. Interestingly, EntNets, which have $> 99\%$ performance when trained with $10K$ examples, only reach 70% (c.f. Henaff et al. [91]) performance with 1K training examples. The Bi-C-GRU model significantly improves on this baseline, which shows that, with less data, coreference annotations can provide a useful bias for a memory network on how to read and write memories.

A break-down of task-wise performance is given in Table 3.7. Comparing C-GRU to the GRU based method, we find that the main gains are on tasks 2 (two supporting facts), 3 (three supporting facts) and 16 (basic induction). All these tasks require aggregation of information across sentences to derive the answer. Comparing to the QRN baseline, we found that C-GRU was significantly worse on task 15 (basic deduction). On closer examination we found that this was because our simplistic coreference module which matches tokens exactly was not able to resolve "mice" to "mouses" and "cats" to "cat". On the other hand, C-GRU was significantly better than QRN on task 16 (basic induction).

We also include a baseline which uses coreference features as 1-hot vectors appended to the input word vectors (GA w/ GRU + 1-hot). This provides the model with information about the coreference clusters, but does not improve performance, suggesting that the regular GRU is unable to track the given coreference information across long distances to solve the task. On the other hand, in Figure 3.7 (left) we show how the performance of GA w/ C-GRU varies as we remove gold-standard mentions from coreference clusters, or if we replace them with random mentions (GA w/ random-GRU). In both cases there is a sharp drop in performance, showing

Task	QRN	GA w/ GRU	GA w/ C-GRU
1: Single Supporting Fact	1.000	0.997	1.000
2: Two Supporting Facts	**0.993**	**0.345**	**0.990**
3: Three Supporting Facts	**0.943**	**0.558**	**0.982**
4: Two Argument Relations	1.000	1.000	1.000
5: Three Argument Relations	0.989	0.989	0.993
6: Yes/No Questions	0.991	0.962	0.976
7: Counting	**0.904**	**0.946**	**0.976**
8: Lists / Sets	0.944	0.947	0.964
9: Simple Negation	1.000	0.991	0.990
10: Indefinite Knowledge	1.000	0.992	0.986
11: Basic Coreference	1.000	0.995	0.996
12: Conjunction	1.000	1.000	0.996
13: Compound Coreference	1.000	0.998	0.993
14: Time Reasoning	**0.992**	**0.895**	**0.849**
15: Basic Deduction	**1.000**	**0.521**	**0.470**
16: Basic Induction	**0.470**	**0.488**	**0.999**
17: Positional Reasoning	0.656	0.580	0.574
18: Size Reasoning	0.921	0.908	0.896
19: Path Finding	0.213	0.095	0.099
20: Agent's Motivation	0.998	0.998	1.000
Average	**0.901**	**0.810**	**0.886**

Table 3.7: Breakdown of task-wise performance on bAbi dataset. Tasks where C-GRU is significant better / worse than either GRU or QRNs are highlighted.

that specifically using coreference for connecting mentions is important.

Wikihop

Dataset. Next we apply our model to the Wikihop dataset [222], which is specifically constructed to test multi-hop reading comprehension across documents. Each instance in this dataset consists of a collection of passages $(p_1, \ldots, p_N)$, and a query of the form (h, r) where h is an entity and r is a relation. The task is to find the tail entity t from a set of provided candidates $\mathcal{C}$.

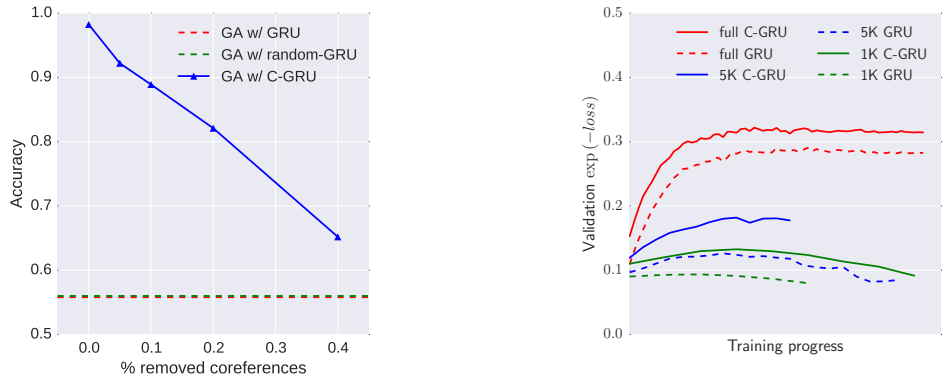


Figure 3.7: (Left) Accuracy of GA w/ C-GRU as coreference annotations are removed for bAbi task 3. (Right) Expected probability of correct answer ($\exp(-loss)$) on Validation set as training progresses on Wikihop dataset for 1K, 5K and the full training datasets.

Implementation. As preprocessing we concatenate all documents in a random order, and extract coreference annotations from the Berkeley Entity Resolution system [62] which gets about 62% F1 score on the CoNLL 2011 test set. We only keep the coreference clusters which contain at least one candidate from $\mathcal{C}$ or an entity which co-occurs with the head entity h. We use a hidden state size of 64, batch size 16, and learning rate of 0.0005 which was halved every 2500 updates. The maximum number of coreference clusters was set to 50 for this dataset. We used dropout of 0.2 in between the intermediate layers, and initialized word embeddings with Glove [155]. We also used character embeddings, which were concatenated with the word embeddings, of size 10. These were output from a CNN layer with 50 filters each of width 5. We also used the qe-feature when processing the documents.

Results. We report results in Table 3.8 when using the full training set, as well as when using a reduced training set of sizes 1K and 5K, to test the model under a low-data regime. In Figure 3.7 we also show the training curves of $\exp(-loss)$ on the validation set. We see higher performance for the C-GRU model in the low data regime, and better generalization throughout the training curve for all three settings. This supports our conjecture that the GRU layer has difficulty learning the kind of coreference-based reasoning required in this dataset, and that the bias towards coreferent recency helps with that. However, perhaps surprisingly, given enough data both models perform comparably. This could either indicate that the baseline learns the required reasoning patterns when given enough data, or, that the bias towards corefence-based reasoning hurts performance for some other types of questions. Indeed, 9% of the questions are

Method	Follow	Follow +single	Follow +multiple	Overall	
	Dev	Dev	Dev	Dev	Test
1K Training Set					
GA w/ GRU	0.307	0.332	0.287	0.263	–
GA w/ C-GRU	0.355	0.370	0.354	0.330	–
5K Training Set					
GA w/ GRU	0.382	0.385	0.390	0.336	–
GA w/ C-GRU	0.452	0.454	0.460	0.401	–
Full Training Set					
BiDAF	–	–	–	–	0.429
GA w/ GRU	0.606	0.615	0.604	0.549	–
GA w/ C-GRU	**0.614**	**0.616**	**0.614**	**0.560**†	**0.593**

Table 3.8: Accuracy on Wikihop for different number of training examples. *Follow:* subset annotated as answer follows from the given passages. *Follow + multiple:* subset annotated as requiring multiple passages for answering. *Follow + single:* subset annotated as requiring one passage for answering. $^{\dagger}p = 0.057$ using Mcnemar's test compared to GA w/ GRU.

answered correctly by the baseline but not by C-GRU, however, we did not find any consistent patterns among these in our analyses.

LAMBADA

Dataset. The LAMBADA dataset [152] is a broad-context language modeling task, consisting of passages from novels, 4-5 sentences long, where the last word needs to be predicted. Interestingly, though, the passages are filtered such that human volunteers were able to predict the missing token given the full passage, but not given only the last sentence. Hence, predicting these tokens involves a broader understanding of the whole passage. Analysis of the questions suggests that around 20% of the questions need coreference understanding to answer correctly [30].

Method	overall	context
Chu et al. [30]	0.4900	–
GA w/ GRU	0.5398	0.6677
GA w/ GRU + 1-hot	0.5338	0.6603
GA w/ C-GRU	**0.5569**	**0.6888**†

Table 3.9: Accuracy on LAMBADA test set, averaged across two runs with random initializations. *context:* passages for which the answer is in context. *overall:* full test set for comparison to prior work. $^{\dagger}p < 0.0001$ using Mcnemar's test compared to GA w/ GRU.

Implementation. We use a hidden state size of 256, batch size of 64, and learning rate of 0.0005 which was halved every 2 epochs. Word vectors were initialized with Glove, and dropout of 0.2 was applied after intermediate layers. The maximum number of coreference clusters in this dataset was 15. We use the same setup as Chu et al. [30] which formulated the problem as a reading comprehension one by treating the last sentence as query, and the remaining passage as context to extract the answer from. In this manner only 80% of the questions are answerable, but the performance increases substantially compared to pure language modeling based approaches. For this dataset we used Stanford CoreNLP to extract coreferences [32], which achieved 0.63 F1 on the CoNLL test set.

Results. Table 3.9 shows a comparison of the GA w/ GRU baseline and GA w/ C-GRU models. We see a significant gain in performance when using the layer with coreference bias. Furthermore, the 1-hot baseline which uses the same coreference information, but with sequential recency bias fails to improve over the regular GRU layer. While the improvement for C-GRU is small, it is significant, and we note that questions in this dataset involve several different types of reasoning out of which we only tackle one specific kind.

3.5 Related Work

3.5.1 Neural Network Readers

Several architectures introduced in Hermann et al. [94] employ LSTM units to compute a combined document-query representation, which is used to rank the candidate answers. These include the *DeepLSTM Reader*, the *Attentive Reader*, and the *Impatient Reader*, each of which use

different methods for mixing representations of the query and document with standard attention mechanisms. The architecture of the Attentive Reader was simplified in *Stanford Attentive Reader*, where shallower recurrent units were used with a bilinear form for the query-document attention [25]. The *Attention-Sum (AS) Reader* [106] further extended this with a scheme named *pointer-sum*, to aggregate final-layer scores of the same entity (Eq. 3.5). Building on the AS Reader, the *Attention-over-Attention (AoA) Reader* [39] introduces a two-way attention mechanism where the query and the document are mutually attentive to each other.

Memory Networks (MemNets) were proposed in Weston et al. [228], where each sentence in the document is encoded to a memory by aggregating nearby words. Attention over the memory slots given the query is used to compute an overall memory and to renew the query representation over multiple iterations, allowing certain types of reasoning over the salient facts in the memory and the query. *Neural Semantic Encoders (NSE)* [145] extended MemNets by introducing a *write* operation which can evolve the memory over time during the course of reading. Iterative reasoning has been found effective in several more recent models, including the *Iterative Attentive Reader* [190] and *ReasoNet* [183]. The latter allows dynamic reasoning steps and is trained with reinforcement learning.

Other related works include *Dynamic Entity Representation network (DER*) [112], which builds dynamic representations of the candidate answers while reading the document, and accumulates the information about an entity by max-pooling; *EpiReader* [207], which consists of two networks, where one proposes a small set of candidate answers, and the other reranks the proposed candidates conditioned on the query and the context; and *Bi-Directional Attention Flow network (BiDAF)* [179], which adopts a multi-stage hierarchical architecture along with a flow-based attention mechanism. Bajgar et al. [8] showed a 10% improvement on the CBT corpus [95] by training the AS Reader on an augmented training set of about 14 million examples, making a case for the community to exploit data abundance.

3.5.2 Linguistic Biases in Deep Learning

An alternative line of work has looked at incorporating linguistic structure into deep learning for NLP. Ji et al. [101] presented a generative model for jointly predicting the next word in the text and its gold-standard coreference annotation. The difference in our work is that we look at the task of reading comprehension, and also work in the more practical setting of system extracted coreferences. *EntNets* [91] also maintain dynamic memory slots for entities, but do not use coreference signals and instead update all memories after reading each sentence, which

leads to poor performance in the low-data regime (c.f. Table 3.6). Here we claim that coreference can provide a useful inductive bias to the model about which memories to use at a particular time-step, and our experiments show a large improvement over EntNets in the low data regime. Yang et al. [248] model references in text as explicit latent variables, but limit their work to text generation. Wang et al. [214] also noted the importance of reference resolution for reading comprehension, and we compare our model to their one-hot pointer reader.

Some works have also used syntax, in the form of dependency trees, to replace the sequential recency bias in RNNs with a syntactic recency bias [27, 160, 198, 199]. However, syntax only looks at dependencies within sentence boundaries, whereas Coref-GRUs focus on longer ranges. Our resulting layer is structurally similar to *GraphLSTMs* [154], with an additional attention mechanism over the graph edges. However, while Peng et al. [154] found that using coreference did not lead to any gains for the task of relation extraction, here we show that it has a positive impact on the reading comprehension task.

3.6 Discussion

In this chapter, we looked at neural architectures for reading comprehension—the task of answering questions about a small passage of text. We presented a multiplicative attention mechanism for modeling interactions between the passage and query, and incorporated it into a multi-layer recurrent neural network model. We also presented an extension of RNNs which incorporates coreference signals when computing text representations, and showed its utility on tasks involving reasoning.

All our experiments were carried out on synthetic data, where either the queries, the context passages, or both were constructed automatically by applying heuristics to some available source of unlabeled data. This was dictated largely by the lack of availability of hand-annotated data at the time this research was carried out. Since then, several large-scale QA datasets have been made publicly available which were constructed either using crowd-sourcing (e.g. Squad [166]), or by harnessing data created for other purposes (e.g. TriviaQA [104], RACE [116] and Natural Questions [115]). Arguably, these datasets capture a more natural distribution of questions that people ask, and do not suffer from some of the biases inherent in synthetic data.[9] While other authors have applied the GA Reader to these and other tasks with favorable results (see, e.g. Lai et al. [116], Yang et al. [244], and Chaplot et al. [23]), the state-of-the-art methods

[9]For example, Kaushik and Lipton [107] showed how in some cases even models trained without either of the query or passage can achieve comparable performance.

today all use variants of Transformer Networks pretrained using large amounts of unlabeled text [51]. We explore one such pre-training strategy in the next chapter.

Jia and Liang [102] showed that, for a wide range of models, adding distractor text to the context passage, which does not answer the query but shares surface similarity with it, can lead to a significant decrease in performance. This brittleness arises, in part, due to the so-called *over-sensitivity* of deep neural networks with millions of parameters, and, in part, due to the purely statistical methods for training them, which are bound to fit to spurious correlations in the data. We expect the models presented in this chapter to also suffer from these limitations, and how to overcome them remains an open research question. Another source of brittleness in the Coref-RNN layer comes from propagation of errors from the external coreference resolution system. As we show in Figure 3.7 (Left), an increase in the error of the coreference annotations results in a decrease in the performance gains due to using them. To our knowledge, other systems which have incorporated linguistic signals into deep learning successfully, e.g. LISA [192], all rely on *gold-standard* annotations at training time. How to best utilize system-extracted signals, a more practical setting, also remains an open question.

A practical limitation with the methods presented in this chapter is that we assume that the gold-standard passage containing the answer is known beforehand. Ultimately, we are interested in open-domain question answering, and instead this must be identified from a large pool of candidate passages, e.g. using a TFIDF based retrieval system. Many works have focused on this setting [26, 31, 55, 219], which presents some new challenges for the reading system, such as deciding when the passage contains an answer and aggregating confidence scores across passages. We explore this setting in more detail in Chapters 5, 7 and 8.

We focused on coreference-based reasoning, which requires aggregating information from multiple mentions of the same entity in a context. Since our work, the Quoref dataset [48] has been constructed specifically to test this kind of reasoning. More generally, reasoning over coreferences is an instance of multi-hop reasoning where multiple facts need to be aggregated in order to answer a query. We will explore multi-hop reasoning in more detail in Chapters 5 and 8. Another important form of reasoning, which we do not tackle here, is numerical operations over text, e.g. as captured in the DROP dataset [61].

Chapter 4

Transfer Learning

The methods presented in the previous chapter, and deep learning models in general, hinge on the availability of large annotated datasets. However, large domain specific annotated datasets are limited and expensive to construct. In this chapter, we explore strategies for transfer learning which exploit unlabeled data to pretrain different parts of the network. First, we focus on how to effectively use pre-trained word embeddings, especially for tokens which are not in the training vocabulary. Next, we introduce a technique which exploits the structure of unlabeled documents to pretrain the entire network. We envision a system where the end user specifies a set of unlabeled documents with a few labelled QA examples over them, and we discuss techniques to maximize reading comprehension performance in such a setting. The work presented in § 4.3 first appeared in the following publication:

- Bhuwan Dhingra, Danish Danish, and Dheeraj Rajagopal. Simple and effective semi-supervised question answering. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 582–587, 2018

4.1 Overview

Many statistical models for tasks related to machine reading employ the following recipe. 1) Tokens in the document and question are represented using word vectors obtained from a lookup table; 2) a text encoder, such as an RNN or Transformer, augmented with an attention mechanism updates these vectors to produce contextual representations; and 3) an output layer uses these contextual representations to classify the start and end of the answer span in the document. The combined number of learnable parameters in these modules is typically on the order

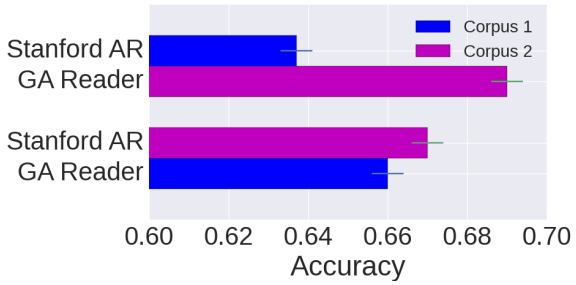


Figure 4.1: Test set accuracies and std error on the Who-Did-What dataset for Stanford AR and GA Reader, trained after initializing with word vectors induced from different corpora. Without controlling for the initialization method, different conclusions may be drawn about which architecture is superior. Corpus 1: BookTest dataset [8], Corpus 2: Wikipedia + Gigaword.

of at least a few million, and hence many labeled examples are required to train them. Unfortunately, practitioners looking to build QA systems for specific applications may not have the resources to collect thousands of questions on corpora of their choice, and these models suffer when training data is limited (c.f. Table 4.3).

Semi-supervised learning methods utilize unlabeled data in addition to labeled training data to improve performance. Here we will discuss methods where the unlabeled data is in the form of text documents (without any associated QA pairs). Specifically, we discuss strategies for transfer learning, where we pretrain parts of the neural network reader on an auxiliary loss function over the unlabeled documents, and then, using the parameter values from this step as initialization, fine-tune the network on the labeled data. These strategies are motivated by the low cost of obtaining a large number of text documents in most domains.

We start by focusing on the first step above, where it is ubiquitous to pretrain the lookup table of word vectors in an unsupervised manner [134, 155]. Through a series of detailed experiments, we study effect of using different pretraining methods, unlabled corpora, embedding dimensions, as well other design choices, on the final reading comprehension performance. We also explore how to best deal with out-of-vocabulary (OOV) tokens, i.e. words which are encountered for the first time during testing. We show that these seemingly minor choices can lead to substantial differences in the final performance of the reader. Furthermore, these differences can be much larger than the gains reported due to architectural improvements. As a concrete example, in Figure 4.1 we compare the performance of two models—Stanford Attentive Reader (AR) [25] and Gated Attention (GA) Reader [54]—on the Who-Did-What dataset [151], initialized with word embeddings trained on different corpora. Clearly, comparison between

architectures is meaningful only under a controlled initialization method.

Next we discuss transfer learning for the other two steps in the model—contextual representations for the text and answer selection. We present a semi-supervised QA system which consists of three stages. First, we construct cloze-style questions (predicting missing spans of text) from the unlabeled corpus; next, we use the generated clozes to pre-train a powerful neural network model for extractive QA [31]; and finally, we fine-tune the model on the small set of provided QA pairs. Our cloze construction process builds on a typical writing phenomenon and document structure: an introduction precedes and summarizes the main body of the article. Many large corpora follow such a structure, including Wikipedia, academic papers, and news articles. We hypothesize that we can benefit from the un-annotated corpora to better answer various questions—at least ones that are lexically similar to the content in base documents and directly require factual information.

We apply the proposed system on three datasets from different domains – SQuAD [166], TriviaQA-Web [104] and the BioASQ challenge [208]. We observe significant improvements in a low-resource setting across all three datasets. For SQuAD and TriviaQA, we attain an F1 score of more than 50% by merely using 1% of the training data. Our system outperforms the approaches for semi-supervised QA presented in Yang et al. [245], and a baseline which uses the same unlabeled data but with a language modeling objective for pretraining. In the BioASQ challenge, we outperform the best performing system from previous year's challenge, improving over a baseline which does transfer learning from the SQuAD dataset. Our analysis reveals that questions which ask for factual information and match to specific parts of the context documents benefit the most from pretraining on automatically constructed clozes.

4.2 Analysis of Word Embeddings

4.2.1 Reading Comprehension Setup

We experiment with two benchmarks from different domains—Who-Did-What (WDW) [151] constructed from news stories, and the Children's Book Test (CBT) [95] constructed from children's books. For CBT we only consider the questions where the answer is a named entity (CBT-NE). Among the numerous models proposed for these benchmarks (see § 3.5 for a list), we pick two – the simple, but competitive, Stanford AR from Chen et al. [25], and the high-performing GA Reader from the previous chapter. The Stanford AR consists of single-layer Bidirectional GRU encoders for both the document and the query, followed by a bilinear atten-

Emb.	Corpus	Domain	Size	Vocab
OTS	Wiki + Gigaword / GoogleNews	Wiki / News	6B / 100B	400K / 3M
WDW	Who-Did-What	News	50M	91K
BT	BookTest	Fiction	8B	1.2M
CBT	Children's BookTest	Fiction	50M	48K

Table 4.1: Details of corpora used for training word embeddings. OTS: Off-The-Shelf embeddings provided with GloVe / word2vec. Corpus size is in # of tokens.

tion operator for computing a weighted average representation of the document. The original model, which was developed for the anonymized CNN / Daily Mail datasets, used an output lookup table W_a to select the answer. However, without anonymization the number of answer candidates can become very large. Hence, we instead select the answer from the document representation itself, followed by the attention-sum mechanism.

For the WDW dataset we use hidden state size $d = 128$ for the GRU and dropout with 0.3 probability. For CBT-NE dataset we use $d = 128$ and dropout with 0.4 probability. The Stanford AR has only 1 layer as proposed in the original paper, while the GA Reader has 3 layers. For Stanford AR dropout is applied to the input of the layer, and for GA Reader it is applied in between layers. Embeddings sizes for the word vectors were set to $d_w = 100$ for all experiments, except those using off-the-shelf word2vec embeddings. To enable a fair comparison, we utilize the qe-feature for Stanford AR, which was used in the implementation of GA Reader. Since our purpose is to study the effect of word vectors, we do not use character embeddings in our experiments. We train the models using the ADAM [109] optimizer with an initial learning rate of 0.0005, which is halved every epoch after the first 3 epochs. We track performance on the validation set, and select the model with the highest validation accuracy for testing.

4.2.2 Pretraining Methods

Two popular methods for inducing word embeddings from text corpora are *GloVe* [155] and *word2vec* [134].[1] These packages also provide off-the-shelf (OTS) embeddings trained on large

[1]This work was carried out before the introduction of ELMo [156] and BERT [51]

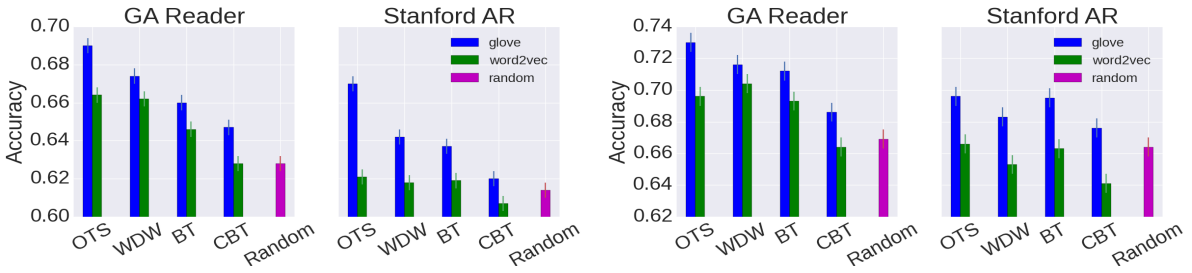


Figure 4.2: Test set accuracy and std error for GA Reader and Stanford AR on WDW (left, middle-left) and CBT-NE (middle-right, right) when trained after initializing with pre-trained embeddings induced from different corpora (Table 4.1), or randomly

corpora.[2] While the GloVe package provides embeddings with varying sizes (50-300), word2vec only provides embeddings of size 300. We also train three additional embeddings, listed in Table 4.1, including those trained on the target datasets themselves. In summary, we test with two in-domain corpora for WDW: one large (OTS) and one small (WDW), and two in-domain corpora for CBT: one large (BT) and one small (CBT).

When training word vectors we retain the default settings provided with the GloVe and word2vec packages, with the only exception that window size was set to 10 for both (to ensure consistency). For word2vec, we used skip-gram architecture with hierarchical softmax, and sub-sampled frequent words with a threshold 10^{-3} (see Mikolov et al. [134] for details). For GloVe, we used 15 iterations when training on the small WDW and CBT corpora, and 50 iterations for the large BT corpus. In any corpus, words occurring less than 5 times were filtered before training the word vectors. Previous studies [121] have shown that hyperparameter choices may have a significant impact on downstream performance. However, training a single RC model can take anywhere from several hours to several days, and tuning hyperparameters for the embedding method on this downstream task is both infeasible and rarely done in practice. Instead, our goal is to provide guidelines to researchers using these methods out-of-the-box.

4.2.3 Performance Comparison

We repeat each experiment twice with different random seeds and report the average test set accuracy across the two runs. Figure 4.2 shows a reading comprehension performance for GA Reader and Stanford AR after initializing with various pre-trained embeddings, and also after

[2]The word2vec package contains embeddings for both capitalized and lowercase words. We convert all words to lowercase, and if a word has both lowercase and uppercase embeddings we use the lowercase version.

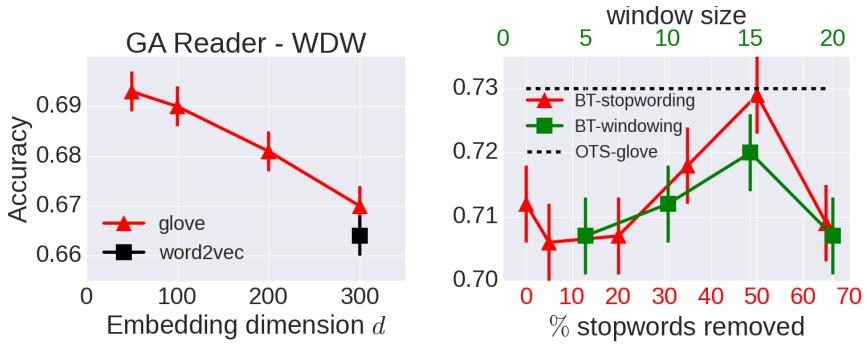


Figure 4.3: Test set accuracy and std error on (left) WDW when initialized with off-the-shelf GloVe embeddings of different sizes, (right) CBT-NE when initialized with embeddings trained on BT corpus after removing a fraction of stopwords (red), or using different window sizes (green).

initializing randomly. We see consistent results across the two datasets and the two models.

The first observation is that using embeddings trained on the right corpora can improve anywhere from 3-6% over random initialization. However, the corpus and method used for pre-training are important choices: for example word2vec embeddings trained on CBT perform worse than random. Also note that in every single case, GloVe embeddings outperform word2vec embeddings trained on the same corpora. It is difficult to claim that one method is better than the other, since previous studies [121] have shown that these methods are sensitive to hyperparameter tuning. However, if used out-of-the-box, GloVe seems to be the preferred method for pre-training.

The single best performance is given by off-the-shelf GloVe embeddings ($d_w = 100$) in each case, which outperform off-the-shelf word2vec embeddings ($d_w = 300$). To understand if the difference comes from the differing dimension sizes, we plot the performance of GloVe embeddings as the dimension size is increased in Figure 4.3 (left). Performance drops as the embedding dimension size is increased (most likely due to over-fitting); however even at $d_w = 300$, GloVe embeddings outperform word2vec embeddings.

On both test datasets embeddings trained on formal domains, like news (OTS, WDW), perform at least as well as those trained on informal ones, like fiction (BT, CBT). This is surprising for CBT-NE dataset which is itself constructed from the informal domain of children's books. For example, WDW (50M tokens) does significantly better than CBT-NE (50M tokens) in 3 out of the 4 cases, and also significantly better than the much larger BT (8B tokens) in one setting (and comparably in other settings). A key distinguishing feature between these two domains is the

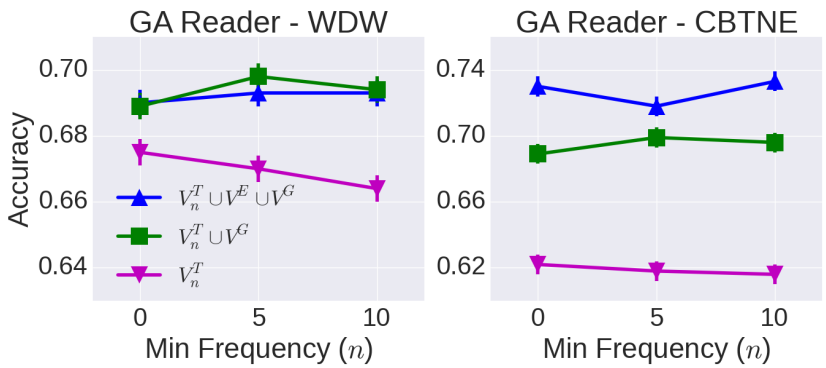


Figure 4.4: Test set accuracy and std error for GA Reader on WDW (left) and CBT-NE (right) when trained after assigning different cuts of the vocabulary with word vectors. *Min frequency* refers to the minimum count of a word type for it to be included in the vocabulary.

fraction of text composed of stopwords – WDW consists of 54% stopwords while BT consists of 68% stopwords. Both GloVe and word2vec induce word vectors by minimizing the Euclidean distance between vectors of frequently co-occurring words. Co-occurrence with stopwords, however, provides little meaningful information about the distributional semantics of a word, and hence corpora with a high percentage of these may not produce high-quality vectors. This effect may be mitigated during pre-training by either, (1) removing a fraction of stopwords from the corpus, (2) increasing the window size for counting co-occuring words. Figure 4.3 (right) shows the effect of both these methods on downstream RC performance. There is an improvement as the fraction of stopwords decreases or the window size increases, upto a certain limit. In fact, with proper tuning, BT embeddings give roughly the same performance as OTS GloVe, emphasizing the importance of hyperparameter tuning when training word vectors. There is also evidence that stopword removal can be beneficial when training word vectors, however this needs further verification on other downstream tasks.

4.2.4 Handling OOV tokens

In this section we study some common techniques for dealing with OOV tokens at test time. Based on the results from the previous section, we conduct this study using only the off-the-shelf GloVe pre-trained embeddings. Let the training and test vocabularies be denoted by $\mathcal{V}^T$ and $\mathcal{V}^E$, respectively, i.e. the collection of all the unique words in the corpus. Further, let $\mathcal{V}^G$ denote the vocabulary of the corpus on which off-the-shelf Glove embeddings are trained. Also define $\mathcal{V}_n^T = \{t \in \mathcal{V}^T : \#t > n\}$ where $\#t$ denotes the count of token t in the training corpus.

Before training a neural network for RC, the developer must first decide on the set of words $\mathcal{V}$ which will be assigned word vectors. Any token outside $\mathcal{V}$ is treated as an OOV token (denoted by UNK) and is assigned the same fixed vector.

By far the most common technique in NLP literature [25, 183] for constructing this vocabulary is to decide on a minimum frequency threshold n (typically 5-10) and set $\mathcal{V} = \mathcal{V}_n^T$. Out of these, vectors for those which also appear in $\mathcal{V}^G$ are initialized to their GloVe embeddings, and the rest are randomly initialized. Remaining tokens in $\mathcal{V}^T$ and those in $\mathcal{V}^E - \mathcal{V}^T$ are all assigned the UNK vector, which is itself updated during training. This method ignores the fact that many of the words assigned as UNK may have already trained embeddings available in $\mathcal{V}^G$. Hence, here we propose another strategy of constructing the vocabulary as $\mathcal{V} = \mathcal{V}_n^T \cup \mathcal{V}^G$. Then at test time, any new token would be assigned its GloVe vector if it exists, or the vector for UNK. A third approach is motivated by the fact that many of the RC models rely on computing a fine-grained similarity between the document and query tokens. Hence, instead of assigning all OOV tokens a common UNK vector, it might be better to assign them untrained but unique random vectors. This can be done by setting the vocabulary to $\mathcal{V} = \mathcal{V}_n^T \cup \mathcal{V}^E \cup \mathcal{V}^G$. Then at test time any new token will be assigned its GloVe vector if it exists, or a random vector. Note that for this approach access to $\mathcal{V}^E$ at training time is not needed.

Figure 4.4 shows a comparison of all three approaches with varying n for the GA Reader on WDW and CBT-NE datasets. A gap of 3% and 11% between the best and worst setting for WDW and CBT-NE respectively clearly indicates the importance of using the correct setting. The commonly used method of setting $\mathcal{V} = \mathcal{V}_n^T$ is not a good choice for RC, and gets worse as n is increased. It performs particularly poorly for the CBT-NE dataset, where $\sim 20\%$ of the test set answers do not appear in the training set (compared to only $\sim 1.5\%$ in WDW). The other two approaches perform comparably for WDW, but for CBT-NE assigning random vectors rather than UNK to OOV tokens gives better performance. This is also easily explained by looking at fraction of test set answers which do not occur in $\mathcal{V}_0^T \cup \mathcal{V}^G$: it is $\sim 10\%$ for CBT-NE, and $< 1\%$ for WDW. Since in general it is not possible to compute these fractions without access to the test set, we recommend setting $\mathcal{V} = \mathcal{V}_n^T \cup \mathcal{V}^E \cup \mathcal{V}^G$.

4.3 Cloze Pretraining

The previous section focused on pretraining the word embedding layer. Here we present a general semi-supervised technique for pretraining the entire reading comprehension architecture.

Passage (P) : Autism is a neurodevelopmental disorder characterized by impaired **social interaction**, verbal and non-verbal communication, and ...restricted and repetitive behavior. Parents usually notice signs in the first two years of their child's life. These signs often develop gradually, though some children with autism reach their developmental milestones at a normal pace and then regress.
Question (Q) : People with autism tend to be a little aloof with little to no _____.
Answer (A) : social interaction

Table 4.2: An example constructed cloze.

4.3.1 System

At a high-level, our system proceeds in three steps. First we generate a set of cloze-style questions from the unlabeled data automatically. Next, we pretrain an existing model to answer these questions. Finally, we fine-tune the model on a small amount of labeled data for the task we care about. We focus on the SQuAD, TriviaQA, and BioASQ datasets in this section (described in detail below).

Cloze Generation. Many web documents follow a template—they begin with an introduction that provides an overview and a brief summary for what is to follow. We assume such a structure while constructing our cloze style questions. When there is no clear demarcation, we treat the first $K\%$ of the document as the introduction (K is a hyperparameter, in our case 20%). While noisy, this heuristic generates a large number of clozes given any corpus, which we found to be beneficial for semi-supervised learning despite the noise.

We use a standard NLP pipeline based on Stanford CoreNLP[3] (for SQuAD, TrivaQA and PubMed) and the BANNER Named Entity Recognizer[4] (only for PubMed articles) to identify entities and phrases. Assume that a document comprises of introduction sentences $\{q_1, q_2, ...q_n\}$, and the remaining passages $\{s_1, s_2, \ldots, s_m\}$. Additionally, let's say that each sentence q_i in introduction is composed of words $\{w_1, w_2, ...w_{l_{q_i}}\}$, where l_{q_i} is the length of q_i. We consider a match(q_i, s_j), if there is an exact string match of a sequence of words $\{w_k, w_{k+1}, ..w_{l_{q_i}}\}$ between the sentence q_i and passage s_j. If this sequence is either a noun phrase, verb phrase,

[3]https://stanfordnlp.github.io/CoreNLP/

[4]http://banner.sourceforge.net

adjective phrase or a named entity in s_j, as recognized by CoreNLP or BANNER, we select it as an answer span a. Additionally, we use s_j as the passage and form a cloze question $\hat{q}_i$ from the answer bearing sentence q_i by replacing a with a placeholder. As a result, we obtain passage-question-answer $(s_j, \hat{q}_i, a)$ triples (Table 4.2 shows an example). As a post-processing step, we prune out (s, q, a) triples where the word overlap between the question (Q) and passage (P) is less than 2 words (after excluding the stop words).

The process relies on the fact that answer candidates from the introduction are likely to be discussed in detail in the remainder of the article. In effect, the cloze question from the introduction and the matching paragraph in the body forms a question and context passage pair. We create two cloze datasets, one each from Wikipedia corpus (for SQuAD and TriviaQA) and PubMed academic papers (for the BioASQ challenge), consisting of 2.2M and 1M clozes respectively. From manually analyzing 100 cloze questions each, we were able to answer 76 from the Wikipedia set and 80 from the PubMed set using the information in the passage. In most cases the cloze paraphrased the information in the passage, which we hypothesized to be a useful signal for the downstream QA task.

We also investigate the utility of forming subsets of the large cloze corpus, where we select the top passage-question-answer triples, based on the different criteria, like i) Jaccard similarity of answer bearing sentence in introduction and the passage ii) the TFIDF scores of answer candidates and iii) the length of answer candidates. However, we empirically find that we were better off using the entire set rather than these subsets.

Pre-training. We make use of the generated cloze dataset to pre-train an expressive neural network designed for the task of reading comprehension. We work with two neural network models—the GA Reader, and BiDAF + Self-Attention (SA) model from Clark and Gardner [31].[5] After pretraining, the performance of BiDAF+SA on a dev set of the (Wikipedia) cloze questions is 0.58 F1 score and 0.55 Exact Match (EM) score. This implies that the cloze corpus is neither too easy, nor too difficult to answer.

Fine Tuning. We fine tune the pre-trained model, from the previous step, over a small set of labeled question-answer pairs. As we shall later see, this step is crucial, and it only requires a handful of labelled questions to achieve a significant proportion of the performance typically attained by training on tens of thousands of questions.

[5] `https://github.com/allenai/document-qa`

4.3.2 Experiments

Datasets. We apply our system to three datasets from different domains. SQuAD [166] consists of questions whose answers are free form spans of text from passages in Wikipedia articles. We follow the same setting as in Yang et al. [245], and split 10% of training questions as the test set, and report performance when training on subsets of the remaining data ranging from 1% to 90% of the full set. We also report the performance on the dev set when trained on the full training set. We compare and study four different settings:

1. *SL*: The Supervised Learning setting, which is only trained on the supervised data.
2. *GDAN*: The best performing model from Yang et al. [245].
3. *LM*: Pretraining on language modeling and fine-tuning on the supervised data.
4. *Cloze*: Pretraining on the Cloze dataset and fine-tuning on the supervised data.

GDAN trains an auxiliary neural network to generate questions from passages by reinforcement learning, and augment the labeled dataset with the generated questions to train the QA model. The LM and Cloze methods use exactly the same data for pretraining, but differ in the loss functions used. We report F1 and EM scores on our test set using the official evaluation scripts provided by the authors of the dataset. Since the GA Reader uses *bidirectional* RNN layers, when pretraining the LM we had to mask the inputs to the intermediate layers partially to avoid the model being exposed to the labels it is predicting. This results in a only a subset of the parameters being pretrained, and leads to a poor performance for this baseline (see Table 4.3). This limitation has since been addressed by Devlin et al. [51], by introducing the *Masked Language Modeling (MLM)* objective (more discussion on this below).

TriviaQA [104] comprises of over 95K web question-answer-evidence triples. Like SQuAD, the answers are spans of text. Similar to the setting in SQuAD, we create multiple smaller subsets of the entire set. For our semi-supervised QA system, we use the BiDAF+SA model [31]—the highest performing publicly available system for TrivaQA. Here again, we compare the supervised learning *SL* settings against the pretraining on *Cloze* set and fine tuning on the supervised set. We report F1 and EM scores on the dev set.[6]

We also test on the BioASQ 5b dataset, which consists of question-answer pairs from PubMed abstracts. We use the publicly available system[7] from Wiese et al. [230], and follow the exact same setup as theirs, focusing only on factoid and list questions. For this setting, there are only

[6]We use a sample of dev questions, which is the default setting for the code by Clark and Gardner [31]. Since our goal is only to compare the models, this is not problematic.

[7]`https://github.com/georgwiese/biomedical-qa`

899 questions for training. Since this is already a low-resource problem we only report results using 5-fold cross-validation on all the available data. We report Mean Reciprocal Rank (MRR) on the factoid questions, and F1 score for the list questions. The reader is referred to Wiese et al. [230] for details.

Model	Method	0		0.01		0.05		0.1		0.2		0.5		0.9		1	
		F1	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1	EM
SQuAD																	
GA	SL	–	–	0.0882	0.0359	0.3517	0.2275	0.4116	0.2752	0.4797	0.3393	0.5705	0.4224	0.6125	0.4684	–	–
GA	GDAN	–	–	–	–	–	–	0.4840	0.3270	0.5394	0.3781	0.5831	0.4267	0.6102	0.4531	–	–
GA	LM	–	–	0.0957	0.0394	0.3141	0.1856	0.3725	0.2365	0.4406	0.2983	0.5111	0.3589	0.5520	0.3964	–	–
GA	Cloze	–	–	0.3090	0.1964	0.4688	0.3385	0.4937	0.3588	0.5575	0.4126	0.6086	0.4679	0.6302	0.4894	–	–
BiDAF+SA	SL	–	–	0.1926	0.1018	0.4764	0.3388	0.5639	0.4258	0.6484	0.5031	0.7044	0.5615	0.7287	0.5874	0.8069	0.7154
BiDAF+SA	Cloze	**0.0682**	**0.032**	**0.5042**	**0.3751**	**0.6324**	**0.4862**	**0.6431**	**0.4995**	**0.6839**	**0.5413**	**0.7151**	**0.5767**	**0.7369**	**0.6005**	**0.8080**	**0.7186**
TRIVIA-QA																	
BiDAF+SA	SL	–	–	0.2533	0.1898	0.4215	0.3566	0.4971	0.4318	0.5624	0.5077	0.6867	0.6239	0.7131	0.6617	0.7291	0.6786
BiDAF+SA	Cloze	**0.1182**	**0.0729**	**0.5521**	**0.4807**	**0.6245**	**0.5614**	**0.6506**	**0.5893**	**0.6849**	**0.6281**	**0.7196**	**0.6607**	**0.7381**	**0.6823**	**0.7461**	**0.6903**

Table 4.3: A holistic view of the performance of our system compared against baseline systems on SQuAD and TriviaQA. Column groups represent different fractions of the training set used for training.

Main Results Table 4.3 shows a comparison of the discussed settings on both SQuAD and TriviaQA. Without any fine-tuning (column 0) the performance is low, probably because the model never saw a real question, but we see significant gains with Cloze pretraining even with very little labeled data. The BiDAF+SA model, exceeds an F1 score of 50% with only 1% of the training data (454 questions for SQuAD, and 746 questions for TriviaQA), and approaches 90% of the best performance with only 10% labeled data. The gains over the SL setting, however, diminish as the size of the labeled set increases and are small when the full dataset is available.

Method	Factoid MRR	List F1
SL*	0.242	0.211
SQuAD pretraining	0.262	0.211
Cloze pretraining	**0.328**	**0.230**

Table 4.4: 5-fold cross-validation results on BioASQ Task 5b. *Our SL experiments showed better performance than what was reported in Wiese et al. [230].

AL	Answer Length
ALP	Answer Location in Passage
ALSP	Answer Location in Sentence
ARC	Answer Rareness w.r.t Cloze corpus
ARS	Answer Rareness w.r.t Squad corpus
ASL	Answer Sentence Length
DL	Document Length
FA	Frequency of Answer in Passage
LOQP	Lexical Overlap Question and Passage
LOQS	Lexical Overlap Question and Answer Sentence
LSQP	Lexical Similarity Question and Passage
LSQS	Lexical Similarity Question and Answer Sentence
PRC	Passage Rareness w.r.t Cloze corpus
PRS	Passage Rareness w.r.t Squad corpus
QL	Question Length
QRC	Question Rareness w.r.t Cloze corpus
QRS	Question Rareness w.r.t Squad corpus

Figure 4.5: Descriptions of the features extracted from the questions.

Cloze pretraining outperforms the GDAN baseline from Yang et al. [245] using the same SQuAD dataset splits. Additionally, we show improvements in the 90% data case unlike GDAN. Our approach is also applicable in the extremely low-resource setting of 1% data, which we suspect GDAN might have trouble with since it uses the labeled data to do reinforcement learning. Furthermore, we are able to use the same cloze dataset to improve performance on both SQuAD and TriviaQA datasets.

On the BioASQ dataset (Table 4.4) we again see a significant improvement when pretraining with the cloze questions over the supervised baseline. The improvement is smaller than what we observe with SQuAD and TriviaQA datasets—we believe this is because questions are generally more difficult in BioASQ. Wiese et al. [230] showed that pretraining on SQuAD dataset improves the downstream performance on BioASQ. Here, we show a much larger improvement by pretraining on cloze questions constructed in an *unsupervised* manner from the same domain.

4.3.3 Analysis

Next, we try to gain an understanding as to which types of questions benefit the most from cloze-pretraining.

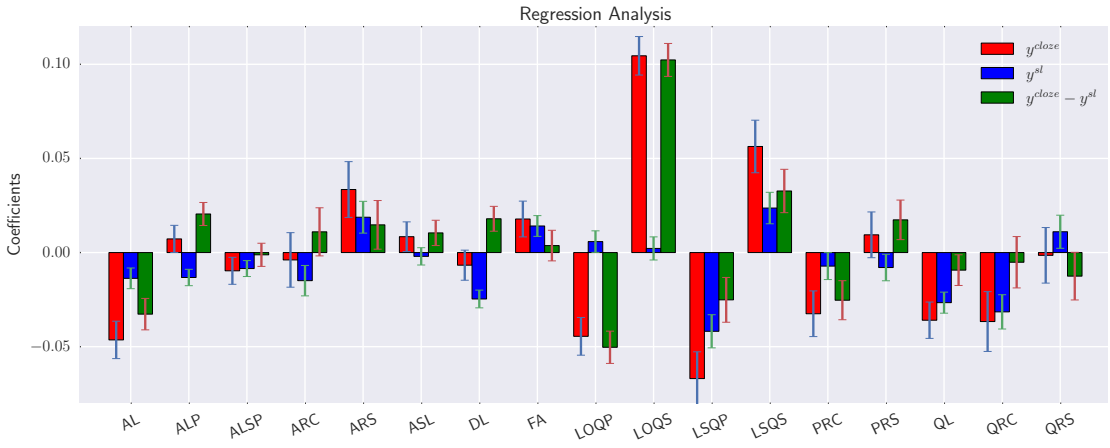


Figure 4.6: Regression coefficients, along with std-errors, when predicting F1 score of *cloze* model, or *sl* model, or the difference of the two, from features computed from SQuAD dev set questions.

Regression Analysis. To understand which types of questions benefit from pre-training, we pre-specified certain features, listed in Figure 4.5, for each of the dev set questions in SQuAD, and then performed linear regression to predict the F1 score for that question from these features. We predict the F1 scores from the cloze pretrained model (y^{cloze}), the supervised model (y^{sl}), and the difference of the two ($y^{\text{cloze}} - y^{\text{sl}}$), when using 10% of labeled data. The coefficients of the fitted model are shown in Figure 4.6 along with their std errors. Positive coefficients indicate that a high value of that feature is predictive of a high F1 score, and a negative coefficient indicates that a small value of that feature is predictive of a high F1 score (or a high difference of F1 scores from the two models in the case of $y^{\text{cloze}} - y^{\text{sl}}$).

The two strongest effects we observe are that a high lexical overlap between the question and the sentence containing the answer is indicative of high boost with pretraining, and that a high lexical overlap between the question and the whole passage is indicative of the opposite. This is hardly surprising, since our cloze construction process is biased towards questions which have a similar phrasing to the answer sentences in context. Hence, test questions with a similar property are answered correctly after pretraining, whereas those with a high overlap with the whole passage tend to have lower performance. The pretraining also favors questions with short answers because the cloze construction process produces short answer spans. Also passages and questions which consist of tokens infrequent in the SQuAD training corpus receive a large boost after pretraining, since the unlabeled data covers a larger domain.

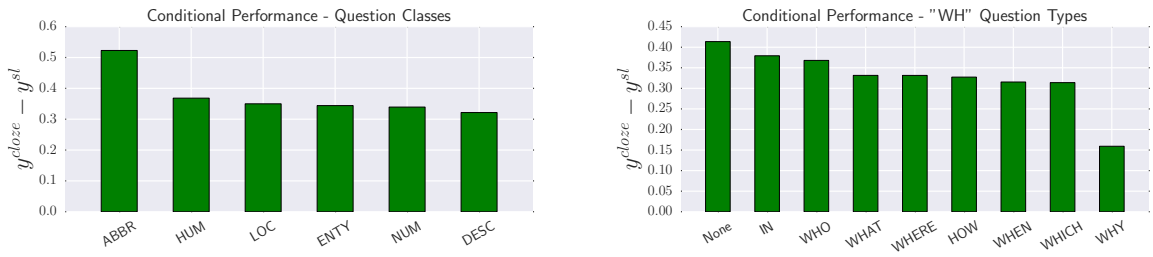


Figure 4.7: Performance gain with pretraining for different subsets of question types.

Performance on question types. Figure 4.7 shows the average gain in F1 score for different types of questions, when we pretrain on the clozes compared to the supervised case. This analysis is done on the 10% split of the SQuAD training set. We consider two classifications of each question – one determined on the first word (usually a wh-word) of the question and one based on the output of a separate question type classifier[8] adapted from [125]. We use the coarse grain labels namely Abbreviation (ABBR), Entity (ENTY), Description (DESC), Human (HUM), Location (LOC), Numeric (NUM) trained on a Logistic Regression classification system . While there is an improvement across the board, we find that abbreviation questions in particular receive a large boost. Also, "why" questions show the least improvement, which is in line with our expectation, since these usually require reasoning or world knowledge which cloze questions rarely require.

4.4 Discussion

In this chapter we looked at techniques for transfer learning from unsupervised data to the task of reading comprehension / QA. Following standard practice at the time, we discussed methods for pre-training the word embedding layer separately from a novel method for pre-training the entire network. However, recent advances have led to a unified framework for pretraining using language modeling which we briefly discuss here.

Peters et al. [156] and McCann et al. [133] proposed *contextualized* word embeddings as an alternative to fixed word embeddings like Glove and word2vec. The former trained two LMs, one from left to right and one from right to left in a text sequence, and used their hidden representations as the contextual embeddings of tokens in a sentence, known as ELMo (Embeddings from Language Models). By replacing regular word embeddings with these, they showed a large improvement on several benchmark NLP tasks, including reading comprehension. The

[8]https://github.com/brmson/question-classification

limitation with ELMo, however, is that it utilizes transfer learning only for the word embedding layer, or the first step noted in the introduction of this chapter. Radford et al. [163] and Devlin et al. [51] proposed neural architectures based on Transformers [209], which could be entirely pretrained using language modeling, and fine-tuned for different tasks with significantly better results than just pretraining the word embeddings.

BERT, from Devlin et al. [51], in particular, addressed an important limitation with how LM pretraining was used in this chapter, by introducing the Masked LM (MLM) objective, which can utilize bidirectional contexts when predicting a token. MLM can be viewed as a combination of the cloze and LM objectives, and proceeds by masking out tokens in the input which are predicted from the context around them. Our cloze-based objective can be viewed as a precursor to BERT, with two main differences. First, our objective masks salient spans (such as named entities) instead of randomly selected tokens. Recent work [86, 105], in fact, has shown that a similar strategy for BERT is also effective. Second, we predict the masked out span using a different context from the same document, in a manner similar to reading comprehension tasks, whereas BERT predicts it from the same context in which the span appears. This further simplifies the construction of the pretraining data, and allows BERT to be trained on a much larger collection of tokens, a key factor in its superior performance.[9]

We also discussed strategies for handling OOV tokens at test time. BERT, and other related models, use WordPiece tokenization [237], which composes representations of rare words from there frequently occurring sub-parts. In this manner, both the embedding of the sub-parts and the composition function are pretrained on the LM objective on large amounts of data. This works well in practice, though a detailed investigation of how the intermediate representations in BERT capture word meanings is missing from the literature.

[9]BERT is trained on 3.3B words, as opposed to the 2.2M clozes we construct here. Each cloze ranges from 1-5 tokens.

Part II

Learning with Knowledge Graphs

Chapter 5

Open-Domain QA

So far we have looked at answering questions against a small given passage. In a practical setting, however, given a question we do not know which passage contains the answer, and instead must search for it in a potentially large corpus. Further, some of the information of interest may be already organized in knowledge bases (KBs), and we would like to utilize these as well in our QA system. Hence, in this chapter, we turn to the practical problem of *open-domain* QA, where, given a question, we need to find its answer in a large knowledge source. Specifically, we will focus on the case where the knowledge source consists of text and a (potentially incomplete) knowledge graph. The work presented in this chapter first appeared in:

- Haitian Sun*, Bhuwan Dhingra*, Manzil Zaheer, Kathryn Mazaitis, Ruslan Salakhutdinov, and William W. Cohen. Open domain question answering using early fusion of knowledge bases and text. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 4231–4242, 2018

Code for reproducing the experiments in this chapter is available on Github.[1]

5.1 Overview

While there has been much interest in training end-to-end deep neural networks for open-domain QA, most existing models answer questions using a single information source: either text from an encyclopedia, or a single KB. Intuitively, the suitability of an information source for QA depends on both its *coverage* and the *difficulty* of extracting answers from it. A large text

[1] `https://github.com/OceanskySun/GraftNet`

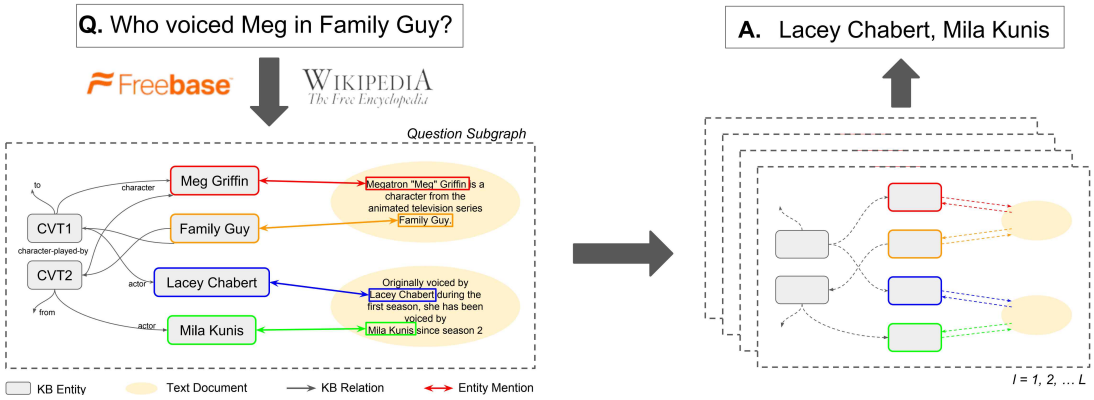


Figure 5.1: (Left) To answer a question posed in natural language, GRAFT-Net considers a heterogeneous graph constructed from text and KB facts, and thus can leverage the rich relational structure between the two information sources. (Right) Embeddings are propagated in the graph for a fixed number of layers (L) and the final node representations are used to classify answers.

corpus has high coverage, but the information is expressed using many different text patterns. As a result, models which operate on these patterns (e.g. the GA Reader from Chapter 3) do not generalize beyond their training domains [58, 229] or to novel types of reasoning [200, 222], and are easily fooled by adversarial inputs [102]. KBs, on the other hand, suffer from low coverage due to their inevitable incompleteness and restricted schema [136], but are easier to extract answers from, since they are constructed precisely for the purpose of being queried.

In practice, some questions are best answered using text, while others are best answered using KBs. A natural question, then, is how to effectively combine both types of information. Surprisingly little prior work has looked at this problem, and in this chapter we focus on a scenario in which a large-scale KB [5, 18] and a text corpus are available, but neither is sufficient alone for answering the questions of interest.

A naïve option, in such a setting, is to take state-of-the-art QA systems developed for each source, and aggregate their predictions using some heuristic [11, 71]. We call this approach *late fusion*, and show that it can be sub-optimal, as models have limited ability to aggregate evidence across the different sources. Instead, we focus on an *early fusion* strategy, where a single model is trained to extract answers from a *question subgraph* (see Fig 5.1, left) containing relevant KB facts as well as text sentences. Early fusion allows more flexibility in combining information from multiple sources.

To enable early fusion, we propose a novel graph convolution based neural network, called

GRAFT-Net (Graphs of Relations Among Facts and Text Networks), specifically designed to operate over heterogeneous graphs of KB facts and text sentences. We build upon recent work on graph representation learning [110, 177], but propose two key modifications to adopt them for the task of QA. First, we propose *heterogeneous update rules* that handle KB nodes differently from the text nodes: for instance, LSTM-based updates are used to propagate information into and out of text nodes. Second, we introduce a *directed propagation method*, inspired by personalized Pagerank in IR [89], which constrains the propagation of embeddings in the graph to follow paths starting from seed nodes linked to the question. Empirically, we show that both these extensions are crucial for the task of QA. An overview of the model is shown in Figure 5.1.

We evaluate these methods on a new suite of benchmark tasks for testing QA models when both KB and text are present. Using WikiMovies [135] and WebQuestionsSP [251], we construct datasets with a varying amount of training supervision and KB completeness, and with a varying degree of question complexity. We report baselines for future comparison, including Key Value Memory Networks [44, 135], and show that our proposed GRAFT-Nets have superior performance across a wide range of conditions. We also show that GRAFT-Nets are competitive with the several high-performing systems developed specifically for the text-only QA or KB-only QA, while being applicable to both.

5.2 Retrieval

Task Description

We assume access to a knowledge base, denoted as $\mathcal{K} = (\mathcal{N}, \mathcal{E}, \mathcal{R})$, where $\mathcal{N}$ is the set of entities in the KB, and the edges $\mathcal{E}$ are triplets (s, r, o) which denote that relation $r \in \mathcal{R}$ holds between the subject $s \in \mathcal{V}$ and object $o \in \mathcal{V}$. We also assume access to a text corpus $\mathcal{S}$, which is a set of documents $\{s_1, \ldots, s_{|\mathcal{S}|}\}$ where each document is a sequence of words $s_i = (w_1, \ldots, w_{|s_i|})$. We further assume that an (imperfect) entity linking system has been run on the collection of documents whose output is a set $\mathcal{L}$ of links (v, s_p) connecting an entity $v \in \mathcal{N}$ with a word at position p in document s, and we denote with $\mathcal{L}_s$ the set of all entity links in document s. For entity mentions spanning multiple words in s, we include links to all the words in the mention in $\mathcal{L}$.

The task is, given a natural language question $q = (w_1, \ldots, w_{|q|})$, extract its answers $\{a\}_q$ from $\mathcal{G} = (\mathcal{K}, \mathcal{S}, \mathcal{L})$. There may be multiple correct answers for a question. Here we will assume that the answers are entities from either the documents or the KB. We are interested in a wide

range of settings for answering the questions, where the KB $\mathcal{K}$ varies from highly incomplete to complete, and we will introduce datasets for testing our models under these settings.

To solve this task we proceed in two steps. First, we extract a subgraph $\mathcal{G}_q \subset \mathcal{G}$ which contains the answer to the question with high probability. The goal for this step is to ensure high recall for answers while producing a graph small enough to fit into GPU memory for gradient-based learning. Next, we use our proposed model GRAFT-Net to learn node representations in $\mathcal{G}_q$, conditioned on q, which are used to classify each node as being an answer or not. Training data for the second step is generated using distant supervision. The entire process mimics the search-and-read paradigm for text-based QA [55].

Question Subgraphs

We retrieve the subgraph $\mathcal{G}_q$ using two parallel pipelines – one over the KB $\mathcal{K}$ which returns a set of entities, and the other over the corpus $\mathcal{S}$ which returns a set of documents. The retrieved entities and documents are then combined with entity links to produce a graph with exactly one connected component.

KB Retrieval. To retrieve relevant entities from the KB we first perform entity linking on the question q, producing a set of *seed entities*, denoted N_q. Next we run the Personalized PageRank (PPR) method (§ 2.2.3) around these seeds to identify other entities which might be an answer to the question. The edge-weights around N_q are distributed equally among all edges of the same type, and they are weighted such that edges relevant to the question receive a higher weight than those which are not. Specifically, we average word vectors to compute a relation vector from the surface form of the relation, and a question vector from the words in the question, and use cosine similarity between these as the edge weights. After running PPR we retain the top E entities $v_1, \ldots, v_E$ by PPR score, along with any edges between them, and add them to $\mathcal{G}_q$.

Text Retrieval. We use Wikipedia as the corpus and retrieve text at the sentence level, i.e. documents in $\mathcal{S}$ are defined along sentences boundaries.[2] We perform text retrieval in two steps: first we retrieve the top 5 most relevant Wikipedia articles, using TFIDF search (§ 2.2.3); then we populate a Lucene[3] index with sentences from these articles, and retrieve the top D sentences $s_1, \ldots, s_D$, based on the words in the question. For the sentence-retrieval step, we found it beneficial to include the title of the article as an additional field in the Lucene index. As

[2]The term *document* will always refer to a sentence in this chapter.

[3]`https://lucene.apache.org/`

most sentences in an article talk about the title entity, this helps in retrieving relevant sentences that do not explicitly mention the entity in the question. We add the retrieved documents, along with any entities linked to them, to the subgraph $\mathcal{G}_q$.

The final question subgraph is $\mathcal{G}_q = (\mathcal{N}_q, \mathcal{E}_q, \mathcal{R}^+)$, where the vertices $\mathcal{N}_q$ consist of all the retrieved entities and documents, i.e. $\mathcal{N}_q = \{v_1, \ldots, v_E\} \cup \{s_1, \ldots, s_D\}$. The edges are all relations from $\mathcal{K}$ among these entities, plus the entity-links between documents and entities, i.e.

$$\mathcal{E}_q = \{(s', o, r) \in \mathcal{E} : s', o \in \mathcal{N}_q, r \in \mathcal{R}\} \cup \{(v, s_p, r_L) : (v, s_p) \in \mathcal{L}_s, s \in \mathcal{V}_q\},$$

where r_L denotes a special "linking" relation. $\mathcal{R}^+ = \mathcal{R} \cup \{r_L\}$ is the set of all edge types in the subgraph.

5.3 GRAFT-Nets

The question q and its answers $\{a\}_q$ induce a labeling of the nodes in $\mathcal{N}_q$: we let $y_v = 1$ if $v \in \{a\}_q$ and $y_v = 0$ otherwise for all $v \in \mathcal{N}_q$. The task of QA then reduces to performing binary classification over the nodes of the graph $\mathcal{G}_q$. In this section we present an extension to the Graph Convolution Network (GCN) model discussed in § 2.2.2 suited for this task.

There are two differences in our setting from previously studied graph-based classification tasks. The first difference is that, in our case, the graph $\mathcal{G}_q$ consists of *heterogeneous* nodes. Some nodes in the graph correspond to KB entities which represent symbolic objects, whereas other nodes represent textual documents which are variable length sequences of words. The second difference is that we want to condition the representation of nodes in the graph on the natural language question q. Below we discuss mechanisms for dealing with both these differences.

Node Initialization

Nodes corresponding to entities are initialized using fixed-size vectors $h_v^{(0)} = x_v \in \mathbb{R}^d$, where x_v can be pre-trained KB embeddings or random, and d is the embedding size. Document nodes in the graph describe a variable length sequence of text. Since multiple entities might link to different positions in the document, we maintain a variable length representation of the document in each layer. This is denoted by $H_s^{(l)} \in \mathbb{R}^{|s| \times d}$. Given the words in the document $(w_1, \ldots, w_{|s|})$, we initialize its hidden representation as:

$$H_s^{(0)} = \text{LSTM}(w_1, w_2, \ldots, w_{|s|}),$$

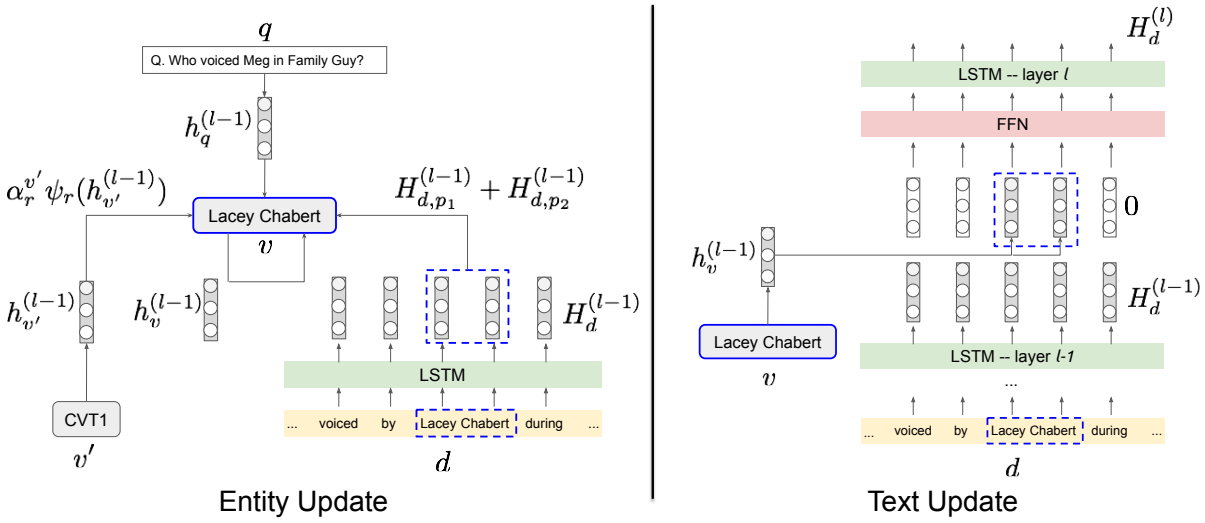


Figure 5.2: Illustration of the heterogeneous update rules for entities (Left) and text documents (Right).

where LSTM refers to a long short-term memory unit. We denote the p-th row of $H_s^{(l)}$, corresponding to the embedding of p-th word in the document s at layer l, as $H_{s,p}^{(l)}$.

Heterogeneous Updates

Node representations in GRAFT-Nets are derived using the same high-level procedure as GCNs, outlined in § 2.2.2, but with different update rules for entities and documents, as illustrated in Figure 5.2 and described below.

Entities. Let $M(v) = \{(s, p)\}$ be the set of positions p in documents s which correspond to a mention of entity v. The update for entity nodes involves a single-layer feed-forward network (FFN) over the concatenation of four states:

$$h_v^{(l)} = \text{FFN}\left(\begin{bmatrix} h_v^{(l-1)} \\ h_q^{(l-1)} \\ \sum_r \sum_{v' \in N_r(v)} \alpha_r^{v'} \psi_r(h_{v'}^{(l-1)}) \\ \sum_{(s,p) \in M(v)} H_{s,p}^{(l-1)} \end{bmatrix}\right). \tag{5.1}$$

The first two terms correspond to the entity representation and question representation (details below), respectively, from the previous layer. The third term aggregates the states from the entity neighbours of the current node, $N_r(v)$, after scaling with an attention weight $\alpha_r^{v'}$ (described in the next section), and applying relation specific transformations ψ_r. The last term aggregates

the states of all tokens that correspond to mentions of the entity v among the documents in the subgraph. Note that the update depends on the positions of entities in their containing document.

Previous work on Relational-GCNs [177] used a linear projection for ψ_r. For a batched implementation, this results in matrices of size $O(B|\mathcal{R}_q||\mathcal{E}_q|d)$, where B is the batch size, which can be prohibitively large for large subgraphs. This is because we have to use adjacency matrices of size $|\mathcal{R}_q| \times |\mathcal{E}_q| \times |\mathcal{E}_q|$ to aggregate embeddings from neighbours of all nodes simultaneously. Hence in this work we use *relation vectors* x_r for $r \in \mathcal{R}_q$ instead of matrices, and compute the update along an edge as:

$$\psi_r(h_{v'}^{(l-1)}) = pr_{v'}^{(l-1)}\text{FFN}\left(x_r, h_{v'}^{(l-1)}\right). \tag{5.2}$$

Here $pr_{v'}^{(l-1)}$ is a PageRank score used to control the propagation of embeddings along paths starting from the seed nodes, which we describe in detail in the next section. The memory complexity of this is $O(B(|\mathcal{F}_q| + |\mathcal{E}_q|)d)$, where $|\mathcal{F}_q|$ is the number of facts in the subgraph $\mathcal{G}_q$.

Documents. Let $L(s, p)$ be the set of all entities linked to the word at position p in document s. The document update proceeds in two steps. First we aggregate over the entity states coming in at each position separately:

$$\tilde{H}_{s,p}^{(l)} = \text{FFN}\left(H_{s,p}^{(l-1)}, \sum_{v \in L(s,p)} h_v^{(l-1)}\right). \tag{5.3a}$$

Here $h_v^{(l-1)}$ are normalized by the number of outgoing edges at v. Next we aggregate states within the document using a unidirectional LSTM:

$$H_s^{(l)} = \text{LSTM}(\tilde{H}_s^{(l)}). \tag{5.3b}$$

Conditioning on the Question

For the parts described thus far, the graph learner is largely agnostic of the question. We introduce dependence on question in two ways: by attention over relations, and by personalized propagation.

To represent q, let $w_1^q, \ldots, w_{|q|}^q$ be the words in the question. The initial representation is computed as:

$$h_q^{(0)} = \text{LSTM}(w_1^q, \ldots, w_{|q|}^q) \in \mathbb{R}^d, \tag{5.4}$$

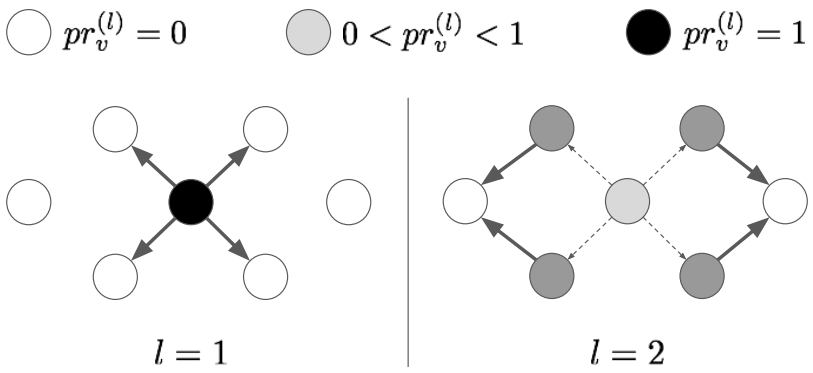


Figure 5.3: Directed propagation of embeddings in GRAFT-Net. A scalar *PageRank* score $pr_v^{(l)}$ is maintained for each node v across layers, which spreads out from the seed node. Embeddings are only propagated from nodes with $pr_v^{(l)} > 0$.

where we extract the final state from the output of the LSTM. In subsequent layers the question representation is updated as $h_q^{(l)} = \text{FFN}\left(\sum_{v \in N_q} h_v^{(l)}\right)$, where N_q denotes the seed entities mentioned in the question.

Attention over Relations. The attention weight in the third term of Eq. 5.1 is computed using the question and relation embeddings:

$$\alpha_r^{v'} = \text{softmax}(x_r^T h_q^{(l-1)}),$$

where the softmax normalization is over all outgoing edges from v', and x_r is the relation vector for relation r. This ensures that embeddings are propagated more along edges relevant to the question.

Directed Propagation. By iteratively aggregating states from the neighbors, the representations learned by our model, and in fact any variant of GCNs, encode *paths* around a node in the graph. Let c be the average in-degree of a node in $\mathcal{G}_q$, then the number of such paths encoded in the representation is c^T, where T is the number of iterations for which the propagation is run. For graphs with both textual and relational edges, c can easily go up to 50-100, and consequently, the number of paths encoded can blow up. This poses a challenge in the setting of learning from denotations since the model can pick up spurious correlations between the question and paths around the answers to that question. To deal with this, we introduce a mechanism for limiting the number of paths encoded in node representations.

Many questions require multi-hop reasoning, which follows a path from a seed node mentioned in the question to the target answer node. To encourage such a behaviour when propagating embeddings, we develop a technique inspired from personalized PageRank (§ 2.2.3). The propagation starts at the seed entities N_q mentioned in the question. In addition to the vector embeddings $h_v^{(l)}$ at the nodes, we also maintain scalar "PageRank" scores $pr_v^{(l)}$ which measure the total weight of paths from a seed entity to the current node, as follows:

$$pr_v^{(0)} = \begin{cases} \frac{1}{|N_q|} & \text{if} \quad v \in N_q, \\ 0 & \text{otherwise,} \end{cases}$$
$$pr_v^{(l)} = (1-\lambda) pr_v^{(l-1)} + \lambda \sum_r \sum_{v' \in N_r(v)} \alpha_r^{v'} pr_{v'}^{(l-1)}.$$

Notice that we reuse the attention weights $\alpha_r^{v'}$ when propagating PageRank, to ensure that nodes along paths relevant to the question receive a high weight. The PageRank score is used as a scaling factor when propagating embeddings along the edges in Eq. 5.2. For $l = 1$, the PageRank score will be 0 for all entities except the seed entities, and hence propagation will only happen outward from these nodes. For $l = 2$, it will be non-zero for the seed entities and their 1-hop neighbors, and propagation will only happen along these edges. Figure 5.3 illustrates this process.

Training & Inference

The final representations $h_v^{(T)} \in \mathbb{R}^d$, are used for binary classification to select the answers:

$$\Pr\left(v \in \{a\}_q | \mathcal{G}_q, q\right) = \sigma(w^T h_v^{(T)} + b), \tag{5.5}$$

where σ is the sigmoid function. Training uses binary cross-entropy loss over these probabilities. To encourage the model to learn a robust classifier, which exploits all available sources of information, we randomly drop KB edges from the graph during training with probability p_0. We call this *fact-dropout*. It is usually easier to extract answers from the KB than from the documents, so the model tends to rely on the former, especially when the KB is complete. This method is similar to DropConnect [213].

Dataset	# train/dev/test	# entities	# relations	# documents	# question words
WikiMovies-10K	10K/10K/10K	43,233	9	79,728	1759
WebQuestionsSP	2848/250/1639	528,617	513	235,567	3781

Table 5.1: Statistics of all the retrieved subgraphs $\cup_q \mathcal{G}_q$ for WikiMovies-10K and WebQuestionsSP.

5.4 Experiments & Results

5.4.1 Datasets

WikiMovies-10K. This dataset consists of 10K randomly sampled training questions from the WikiMovies dataset [135], along with the original test and validation sets. We sample the training questions to create a more difficult setting, since the original dataset has 100K questions over only 8 different relation types, which is unrealistic in our opinion. In § 5.4.2 we also compare to the existing state-of-the-art using the full training set.

We use the KB and text corpus constructed from Wikipedia released by Miller et al. [135]. For entity linking we use simple surface level matches, and retrieve the top 50 entities around the seeds to create the question subgraph. We further add the top 50 sentences (along with their article titles) to the subgraph using Lucene search over the text corpus. The overall answer recall in our constructed subgraphs is 99.6%.

WebQuestionsSP [251]. This dataset consists of 4737 natural language questions posed over Freebase [18] entities, split up into 3098 training and 1639 test questions. We reserve 250 training questions for model development and early stopping. We use the entity linking outputs from S-MART[4] and retrieve 500 entities from the neighbourhood around the question seeds in Freebase to populate the question subgraphs.[5] We further retrieve the top 50 sentences from Wikipedia with the two-stage process described earlier. The overall recall of answers among the subgraphs is 94.0%.

Table 5.1 shows the combined statistics of all the retreived subgraphs for the questions in each dataset. These two datasets present varying levels of difficulty. While all questions in WikiMovies correspond to a single KB relation, for WebQuestionsSP the model needs to aggre-

[4] https://github.com/scottyih/STAGG

[5] A total of 13 questions had no detected entities. These were ignored during training and considered as incorrect during evaluation.

Model	Text Only	KB + Text			
		10 %	30%	50%	100%
WikiMovies-10K					
KV-KB	–	15.8 / 9.8	44.7 / 30.4	63.8 / 46.4	94.3 / 76.1
KV-EF	50.4 / 40.9	53.6 / 44.0	60.6 / 48.1	75.3 / 59.1	93.8 / 81.4
GN-KB	–	19.7 / 17.3	48.4 / 37.1	67.7 / 58.1	**97.0 / 97.6**
GN-LF	**73.2 / 64.0**	74.5 / 65.4	78.7 / 68.5	83.3 / 74.2	96.5 / 92.0
GN-EF		75.4 / 66.3	82.6 / 71.3	87.6 / 76.2	96.9 / 94.1
GN-EF+LF		**79.0 / 66.7**	**84.6 / 74.2**	**88.4 / 78.6**	**96.8 / 97.3**
WebQuestionsSP					
KV-KB	–	12.5 / 4.3	25.8 / 13.8	33.3 / 21.3	46.7 / 38.6
KV-EF	23.2 / 13.0	24.6 / 14.4	27.0 / 17.7	32.5 / 23.6	40.5 / 30.9
GN-KB	–	15.5 / 6.5	34.9 / 20.4	47.7 / 34.3	66.7 / 62.4
GN-LF	**25.3 / 15.3**	29.8 / 17.0	39.1 / 25.9	46.2 / 35.6	65.4 / 56.8
GN-EF		31.5 / 17.7	40.7 / 25.2	49.9 / 34.7	67.8 / 60.4
GN-EF+LF		**33.3 / 19.3**	**42.5 / 26.7**	**52.3 / 37.4**	**68.7 / 62.3**

Table 5.2: Hits@1 / F1 scores of GRAFT-Nets (GN) compared to KV-MemNN (KV) in KB only (-KB), early fusion (-EF), and late fusion (-LF) settings.

gate over two KB facts for $\sim$30% of the questions, and also requires reasoning over constraints for $\sim$7% of the questions [129]. For maximum portability, QA systems need to be robust across several degrees of KB availability since different domains might contain different amounts of structured data; and KB completeness may also vary over time. Hence, we construct an additional 3 datasets each from the above two, with the number of KB facts downsampled to 10%, 30% and 50% of the original to simulate settings where the KB is incomplete. We repeat the retrieval process for each sampled KB.

5.4.2 Main Results

Compared Models

Our main baseline is the work of Das et al. [44], which attempts an early fusion strategy for QA over KB facts and text. Their approach is based on Key-Value Memory Networks (KV-

MemNNs) [135] coupled with a universal schema [169] to populate a memory module with representations of KB triples and text snippets independently. The key limitation for this model is that it ignores the rich relational structure between the facts and text snippets. We compare the following variants of their model and ours:

- **KV-KB** is the Key Value Memory Networks model but using only KB and ignoring the text.
- **KV-EF** (early fusion) is the same model with access to both KB and text as memories. For text we use a BiLSTM over the entire sentence as keys, and entity mentions as values. This re-implementation shows better performance on the text-only and KB-only WikiMovies tasks than the results reported previously (see Table 5.3).[6]
- **GN-KB** is the GRAFT-Net model ignoring the text.
- **GN-LF** is a late fusion version of the GRAFT-Net model: we train two separate models, one using text only and the other using KB only, and then ensemble the two.[7]
- **GN-EF** is our main GRAFT-Net model with early fusion.
- **GN-EF+LF** is an ensemble over the GN-EF and GN-LF models, with the same ensembling method as GN-LF.

We report Hits@1, which is the accuracy of the top-predicted answer from the model, and the F1 score. To compute the F1 score we tune a threshold on the development set to select answers based on binary probabilities for each node in the subgraph.

Performance

Table 5.2 presents a comparison of the above models across all datasets. GRAFT-Nets (GN) shows consistent improvement over KV-MemNNs on both datasets in all settings, including KB only (-KB), text only (-EF, Text Only column), and early fusion (-EF). Interestingly, we observe a larger relative gap between the Hits and F1 scores for the KV models than we do for our GN models. This is because the attention for KV is normalized over the memories, which are KB facts (or text sentences): hence the model is unable to assign high probabilities to multiple facts at the same time. On the other hand, in GN, we normalize the attention over *types of relations* outgoing from a node, and hence can assign high weights to all the correct answers.

[6]For all KV models we tuned the number of layers $\{1, 2, 3\}$, batch size $\{10, 30, 50\}$, model dimension $\{50, 80\}$. We also use fact dropout regularization in the KB+Text setting tuned between $\{0, 0.2, 0.4\}$.

[7]For ensembles we take a weighted combination of the answer probabilities produced by the models, with the weights tuned on the dev set. For answers only in text or only in KB, we use the probability as is.

Method	WikiMovies (full)		WebQuestionsSP	
	kb	doc	kb	doc
MINERVA	**97.0 / –**	–	–	–
R2-AsV	–	85.8 / –	–	–
NSM	–	–	**– / 69.0**	–
DrQA*	–	–	–	21.5 / –
R-GCN#	96.5 / 97.4	–	37.2 / 30.5	–
KV	93.9 / –	76.2 / –	– / –	– / –
KV#	95.6 / 88.0	80.3 / 72.1	46.7 / 38.6	23.2 / 13.0
GN	96.8 / 97.2	**86.6 / 80.8**	**67.8** / 62.8	**25.3 / 15.3**

Table 5.3: Hits@1 / F1 scores compared to SOTA models using only KB or text: MINERVA [42], R2-AsV [221], Neural Symbolic Machines (NSM) [129], DrQA [26], R-GCN [177] and KV-MemNN [135]. *DrQA is pretrained on SQuAD. #Re-implemented.

We also see a consistent improvement of early fusion over late fusion (-LF), and by ensembling them together we see the best performance across all the models. In Table 5.2 (right), we further show the improvement for KV-EF over KV-KB, and GN-LF and GN-EF over GN-KB, as the amount of KB is increased. This measures how effective these approaches are in utilizing text plus a KB. For KV-EF we see improvements when the KB is highly incomplete, but in the full KB setting, the performance of the fused approach is worse. A similar trend holds for GN-LF. On the other hand, GN-EF with text improves over the KB-only approach in all settings. As we would expect, though, the benefit of adding text decreases as the KB becomes more and more complete.

Comparison to Specialized Methods

In Table 5.3 we compare GRAFT-Nets to state-of-the-art models that are specifically designed and tuned for QA using either only KB or only text. For this experiment we use the full WikiMovies dataset to enable direct comparison to previously reported numbers. For DrQA [26], following the original paper, we restrict answer spans for WebQuestionsSP to match an entity in Freebase. In each case we also train GRAFT-Nets using only KB facts or only text sentences. In three out of the four cases, we find that GRAFT-Nets either match or outperform the existing state-of-the-art models. We emphasize that the latter have no mechanism for dealing with the fused setting.

Question	Correct Answers	Predicted Answers
what language do most people speak in afghanistan	Pashto language, Farsi (Eastern Language)	Pashto language
what college did john stockton go to	Gonzaga University	Gonzaga University, Gonzaga Preparatory School

Table 5.4: Examples from WebQuestionsSP dataset. Top: The model misses a correct answer. Bottom: The model predicts an extra incorrect answer.

The one exception is the KB-only case for WebQuestionsSP where GRAFT-Net does 6.2% F1 points worse than Neural Symbolic Machines [129]. Analysis suggested three explanations: (1) In the KB-only setting, the recall of subgraph retrieval is only 90.2%, which limits overall performance. In an oracle setting where we ensure the answers are part of the subgraph, the F1 score increases by 4.8%. (2) We use the same probability threshold for all questions, even though the number of answers may vary significantly. Models which parse the query into a symbolic form do not suffer from this problem since answers are retrieved in a deterministic fashion. If we tune separate thresholds for each question the F1 score improves by 7.6%. (3) GRAFT-Nets perform poorly in the few cases where there is a constraint involved in picking out the answer (for example, "who *first* voiced Meg in Family Guy"). If we ignore such constraints, and consider all entities with the same sequence of relations to the seed as correct, the performance improves by 3.8% F1. Heuristics such as those used by Yu et al. [256] can be used to improve these cases. Figure 5.4 shows examples where GRAFT-Net fails to predict the correct answer set exactly.

5.4.3 Analysis

Heterogeneous Updates. We tested a non-heterogeneous version of our model, where instead of using fine-grained entity linking information for updating the node representations ($M(v)$ and $L(s,p)$ in Eqs. 5.1 & 5.3a), we aggregate the document states across all its positions as $\sum_p H^{(l)}_{s,p}$ and use this combined state for all updates. Without the heterogeneous update, all entities $v \in L(s,\cdot)$ will receive the same update from document s. Therefore, the model cannot disambiguate different entities mentioned in the same document. The result in Table 5.5 shows that this version is consistently worse than the heterogeneous model.

	0 KB	0.1 KB	0.3 KB	0.5 KB	1.0 KB
NH	22.7 / 13.6	28.7 / 15.8	35.6 / 23.2	47.2 / 33.3	66.5 / 59.8
H	25.3 / 15.3	31.5 / 17.7	40.7 / 25.2	49.9 / 34.7	67.8 / 60.4

Table 5.5: Non-Heterogeneous (NH) vs. Heterogeneous (H) updates on WebQuestionsSP

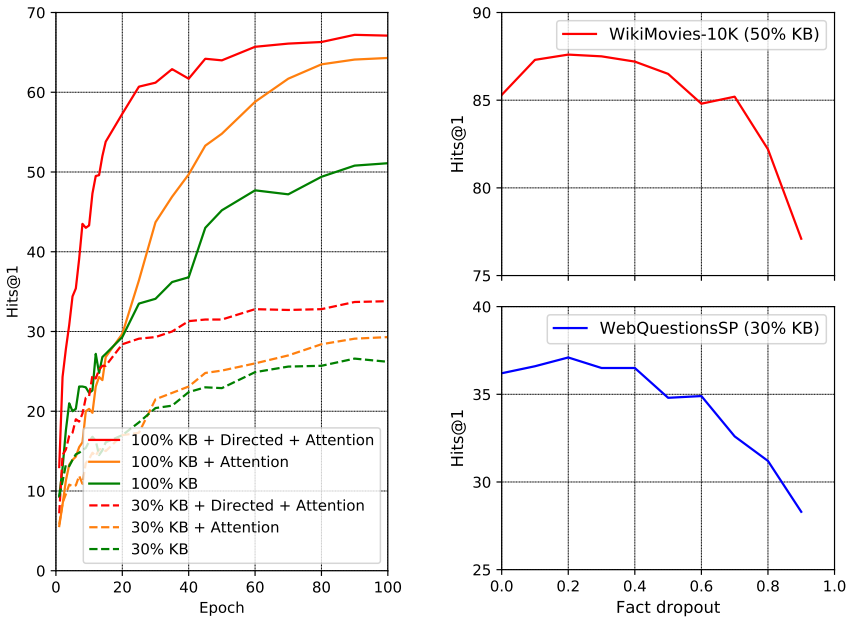


Figure 5.4: Left: Effect of directed propagation and query-based attention over relations for the WebQuestionsSP dataset with 30% KB and 100% KB. Right: Hits@1 with different rates of fact-dropout on and WikiMovies and WebQuestionsSP.

Conditioning on the Question. We performed an ablation test on the directed propagation method and attention over relations. We observe that both components lead to better performance. Such effects are observed in both complete and incomplete KB scenarios, e.g. on WebQuestionsSP dataset, as shown in Figure 5.4 (left).

Fact Dropout. Figure 5.4 (right) compares the performance of the early fusion model as we vary the rate of fact dropout. Moderate levels of fact dropout improve performance on both datasets. The performance increases as the fact dropout rate increases until the model is unable to learn the inference chain from KB.

5.5 Discussion

In this chapter we investigated QA using text combined with an incomplete KB, a task which has received limited attention in the past. We discussed two broad approaches to solving this problem—"late fusion" and "early fusion", and showed that early fusion approaches perform better. We also introduced a novel early-fusion model, called GRAFT-Net, for classifying nodes in subgraph consisting of both KB entities and text documents. GRAFT-Net builds on recent advances in graph representation learning but includes several innovations which improve performance on this task. GRAFT-Nets are a single model which achieve performance competitive to state-of-the-art methods in both text-only and KB-only settings, and outperform baseline models when using text combined with an incomplete KB.

There are, however, limitations with the approach. First, GRAFT-Nets are limited to questions whose answers are entities in the KB. In principle, this can be easily extended to include all spans in the text, using a similar mechanism to that used in Chapter 3, but it is unclear how this will affect model performance. Secondly, and more importantly, the subgraph retrieval process, which relies on heuristics and an external entity linking system, is error prone and limits the overall performance. This is particularly limiting when dealing with multi-hop questions whose answers may lie across passages—a finding observed by Sun et al. [197].

PullNet [197] and CogQA [60] are two models which build on our work to deal with the second limitation above. Both of these iteratively expand the question sub-graph, starting from only the entities in the question (N_q here) by learning which entity nodes to expand based on the question at each step. In the end a similar classification step to ours is used to identify the answer. PullNet, in particular, improves on our approach when dealing with multi-hop questions. A general theme used in these works for improving passage retrieval is that connections between different passages in a text corpus can be modeled using entity links to a KB [138]. We will discuss this idea in more detail in Chapter 8.

As LM pretraining is becoming increasingly popular for NLP tasks, one line of work has also looked at extending prediction in state-of-the-art LMs with facts from KBs [132, 157], or knowledge from text [86]. A broader research agenda is concerned with connecting KBs with deep learning for various NLP tasks, e.g. information extraction [240], machine translation [144], and natural language inference [220]. In the next chapter, we present a method for utilizing KBs with reinforcement learning for dialogue.

Chapter 6

Multi-turn QA

The previous chapter focused on the setting where we learn to answer a single fixed query from the user. Implicitly, this assumes that the user will provide a coherent, well-formed and complete query to the system in one pass. In practice, for complex information needs, users may prefer a sequence of short and simple queries, each of which may be incomplete, but together specify a unique answer. Hence, in this chapter we focus on a multi-turn QA, where the user starts with an under-specified query, and the agent responds with follow-up questions to precisely locate the information requested. We will focus on a reinforcement learning framework for this purpose, and make the simplifying assumption that the underlying knowledge source is a small entity-centric KB, described below. This work first appeared in:

- Bhuwan Dhingra, Lihong Li, Xiujun Li, Jianfeng Gao, Yun-Nung Chen, Faisal Ahmed, and Li Deng. Towards end-to-end reinforcement learning of dialogue agents for information access. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 484–495, 2017

Code for the experiments in this chapter is available on Github.[1]

6.1 Overview

We focus on KB-InfoBots, a dialogue agent that helps users navigate a KB in search of an entity, as illustrated by the example in Figure 6.1. Conventional dialogue systems are composed of several modules, which perform the functions of natural language understanding, tracking

[1] `https://github.com/MiuLab/KB-InfoBot`

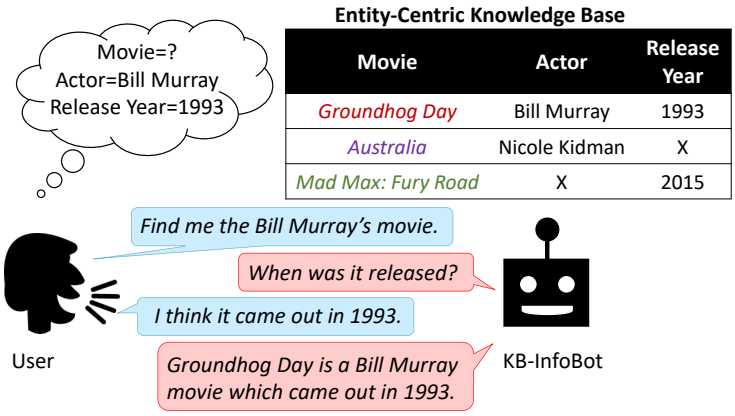


Figure 6.1: An interaction between a user looking for a movie and the KB-InfoBot. An entity-centric knowledge base is shown above the KB-InfoBot (missing values denoted by X).

user beliefs, formulating a dialogue policy and generating natural language. Reinforcement Learning (RL) has been explored to leverage user interactions so train all these components [74, 123, 225], which can potentially lead to systems that adapt with time. At the same time, there is also interest in end-to-end dialogue systems, which combine feature extraction and policy optimization using deep neural networks.

For task-oriented dialogue applications, as is the case here, during the interaction the agent must also query a KB to retrieve information, and use that information to formulate its response to the user. One approach for doing so is by performing semantic parsing on the input to construct a symbolic query representing the beliefs of the agent about the user goal. We call such an operation a *Hard-KB* lookup. This approach has two drawbacks: (1) the retrieved results do not carry any information about uncertainty in semantic parsing, and (2) the retrieval operation is non differentiable, and hence the parser and dialog policy are trained separately. This makes online end-to-end learning from user feedback difficult once the system is deployed. For example, Wen et al. [225] introduced a modular neural dialogue agent, which used a Hard-KB lookup, and hence required separate labeled data to train the different modules.

Here we propose a probabilistic framework for computing the posterior distribution of the user target over a knowledge base, which we term a *Soft-KB* lookup (§ 6.2). This distribution is constructed from the agent's belief about the attributes of the entity being searched for. The dialogue policy network, which decides the next system action, receives as input this full distribution instead of a handful of retrieved results. We show in our experiments that this framework allows the agent to achieve a higher task success rate in fewer dialogue turns. Further, the

retrieval process is differentiable, allowing us to construct an end-to-end trainable KB-InfoBot, all of whose components are updated online using RL.

Reinforcement learners typically require an environment to interact with, and hence static dialogue corpora cannot be used for their training. Running experiments on human subjects, on the other hand, unfortunately is too expensive. A common workaround in the dialogue community [175, 176, 254] is to instead use *user simulators* which mimic the behavior of real users in a consistent manner. For training KB-InfoBot, we adapt the publicly available[2] simulator described in Li et al. [126]. We evaluate several versions of KB-InfoBot with the simulator and on real users, and show that the proposed Soft-KB lookup helps the reinforcement learner discover better dialogue policies. Initial experiments on the end-to-end agent also demonstrate its strong learning capability.

6.2 Probabilistic KB Lookup

We first list our assumptions about the underlying knowledge base that the dialogue agent can access, and then describe a probabilistic framework for querying the KB given the agent's beliefs over the fields in the KB.

Entity-Centric Knowledge Base (EC-KB)

In § 2.1.2 we introduced KBs as a collection of triples of the form (s, r, o), where s is the subject, r the relation, and o the object. For KB-InfoBot, we further assume that this is *entity-centric* (EC-KB) [262], which means that all subjects are of a single type (such as movies or persons), and the relations correspond to attributes of these head entities. Such KBs are quite common, and can be converted to a table format whose rows correspond to the unique head entities, columns correspond to the unique relation types (*slots* henceforth), and some entries may be missing. An example is shown in Figure 6.1.

Notations and Assumptions

Let $\mathcal{T}$ denote the KB table described above and $\mathcal{T}_{i,j}$ denote the jth slot-value of the ith entity. $1 \leq i \leq N$ and $1 \leq j \leq M$. We let $\mathcal{V}^j$ denote the vocabulary of each slot, i.e. the set of all distinct values in the j-th column. We denote missing values from the table with a special token and write $\mathcal{T}_{i,j} = \Psi$. $M_j = \{i : \mathcal{T}_{i,j} = \Psi\}$ denotes the set of entities for which the value

[2] https://github.com/MiuLab/TC-Bot

of slot j is missing. Note that the user may still know the actual value of $\mathcal{T}_{i,j}$, and we assume this lies in $\mathcal{V}^j$. We do not deal with new entities or relations at test time.

We assume a uniform prior $G \sim \mathcal{U}[\{1, ...N\}]$ over the rows in the table $\mathcal{T}$, and let binary random variables $\Phi_j \in \{0, 1\}$ indicate whether the user knows the value of slot j or not. The agent maintains M multinomial distributions $p_j^t(v)$ for $v \in \mathcal{V}^j$ denoting the probability at turn t that the user constraint for slot j is v, given their utterances U_1^t till that turn. The agent also maintains M binomials $q_j^t = \Pr(\Phi_j = 1)$ which denote the probability that the user knows the value of slot j.

We assume that column values are independently distributed to each other. This is a strong assumption but it allows us to model the user goal for each slot independently, as opposed to modeling the user goal over KB entities directly. Typically $\max_j |\mathcal{V}^j| < N$ and hence this assumption reduces the number of parameters in the model.

Soft-KB Lookup

Let $p_{\mathcal{T}}^t(i) = \Pr(G = i|U_1^t)$ be the posterior probability that the user is interested in row i of the table, given the utterances up to turn t. We assume all probabilities are conditioned on user inputs U_1^t and drop it from the notation below. From our assumption of independence of slot values, we can write $p_{\mathcal{T}}^t(i) \propto \prod_{j=1}^{M} \Pr(G_j = i)$, where $\Pr(G_j = i)$ denotes the posterior probability of user goal for slot j pointing to $\mathcal{T}_{i,j}$. Marginalizing this over Φ_j gives:

$$\begin{aligned}\Pr(G_j = i) &= \sum_{\phi=0}^{1} \Pr(G_j = i, \Phi_j = \phi) \\ &= q_j^t \Pr(G_j = i|\Phi_j = 1) + (1 - q_j^t) \Pr(G_j = i|\Phi_j = 0).\end{aligned} \tag{6.1}$$

For $\Phi_j = 0$, the user does not know the value of the slot, and from the prior:

$$\Pr(G_j = i|\Phi_j = 0) = \frac{1}{N}, \quad 1 \leq i \leq N. \tag{6.2}$$

For $\Phi_j = 1$, the user knows the value of slot j, but this may be missing from $\mathcal{T}$, and we again have two cases. For the case when $i \in M_j$, we can write (dropping the condition on $\Phi_j = 1$ for brevity):

$$\begin{aligned}\Pr(G_j = i) &= \Pr(G_j \in M_j) \Pr(G_j = i|G_j \in M_j) \\ &= \frac{|M_j|}{N} \frac{1}{|M_j|} \\ &= \frac{1}{N},\end{aligned} \tag{6.3}$$

where we assume all missing values to be equally likely, and estimate the prior probability of the goal being missing from the count of missing values in that slot. For the case when $i = v \notin M_j$:

$$\begin{aligned}\Pr(G_j = i) &= \Pr(G_j \notin M_j) \Pr(G_j = i | G_j \notin M_j) \\ &= \left(1 - \frac{|M_j|}{N}\right) \times \frac{p_j^t(v)}{N_j(v)},\end{aligned} \tag{6.4}$$

where the second term comes from taking the probability mass associated with v in the belief tracker and dividing it equally among all rows with value v. Here, $N_j(v)$ is the count of value v in slot j. Combining the above, we get:

$$\Pr(G_j = i | \Phi_j = 1) = \begin{cases} \frac{1}{N}, & i \in M_j \\ \frac{p_j^t(v)}{N_j(v)}\left(1 - \frac{|M_j|}{N}\right), & i \notin M_j \end{cases} \tag{6.5}$$

Combining Eqs. 6.1, 6.2, and 6.5 gives us the procedure for computing the posterior over KB entities.

6.3 KB-InfoBot

We claim that the Soft-KB lookup method has two benefits over the Hard-KB method. First, it helps the agent discover better dialogue policies by providing it more information from the language understanding unit. Second, it allows end-to-end training of both dialogue policy and language understanding in an online setting. In this section we describe several agents to test these claims.

Overview

Figure 6.2 shows an overview of the components of the KB-InfoBot. At each turn, the agent receives a natural language utterance u^t as input, and selects an action a^t as output. The action space, denoted by $\mathcal{A}$, consists of $M + 1$ actions — *request(slot=i)* for $1 \leq i \leq M$ will ask the user for the value of slot i, and *inform(I)* will inform the user with an ordered list of results I from the KB. The dialogue ends once the agent chooses *inform*.

We adopt a modular approach, typical to goal-oriented dialogue systems [225], consisting of: a *belief tracker* module for identifying user intents, extracting associated slots, and tracking the dialogue state [28, 87, 92, 93, 249]; an interface with the database to query for relevant results (*Soft-KB* lookup); a *summary* module to summarize the state into a vector; and a *dialogue*

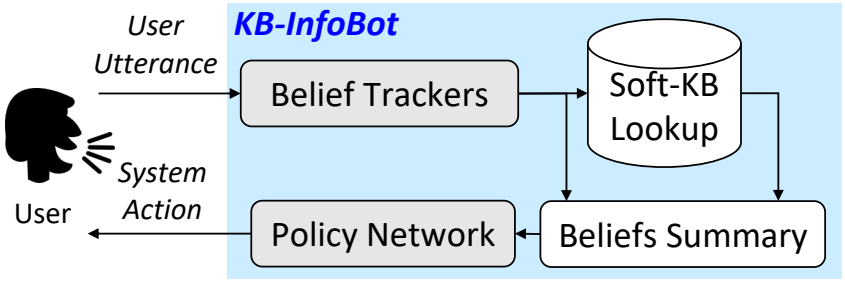


Figure 6.2: High-level overview of the end-to-end KB-InfoBot. Components with trainable parameters are highlighted in gray.

policy which selects the next system action based on current state [254]. We assume the agent only responds with dialogue acts. A template-based *Natural Language Generator* (NLG) can be constructed for converting dialogue acts into natural language.

Belief Trackers

The InfoBot consists of M belief trackers, one for each slot, which get the user input x^t and produce two outputs, p_j^t and q_j^t, which we shall collectively call the *belief state*: p_j^t is a multinomial distribution over the slot values v, and q_j^t is a scalar probability of the user knowing the value of slot j. We describe two versions of the belief tracker.

Hand-Crafted Tracker. We first identify mentions of slot-names (such as "actor") or slot-values (such as "Bill Murray") from the user input u^t, using token-level keyword search. Let $\{w \in x\}$ denote the set of tokens in a string x[3], then for each slot in $1 \leq j \leq M$ and each value $v \in \mathcal{V}^j$, we compute its *matching score* as follows:

$$s_j^t[v] = \frac{|\{w \in u^t\} \cap \{w \in v\}|}{|\{w \in v\}|} \tag{6.6}$$

A similar score b_j^t is computed for the slot-names. A one-hot vector $req^t \in \{0,1\}^M$ denotes the previously requested slot from the agent, if any. q_j^t is set to 0 if $req^t[j]$ is 1 but $s_j^t[v] = 0$ $\forall v \in \mathcal{V}^j$, i.e. the agent requested for a slot but did not receive a valid value in return, else it is set to 1.

Starting from an prior distribution p_j^0 (based on the counts of the values in the KB), $p_j^t[v]$ is updated as:

$$p_j^t[v] \propto p_j^{t-1}[v] + C\left(s_j^t[v] + b_j^t + \mathbb{1}(req^t[j] = 1)\right) \tag{6.7}$$

[3]We use the NLTK tokenizer available at `http://www.nltk.org/api/nltk.tokenize.html`

Here C is a tuning parameter, and the normalization is given by setting the sum over v to 1.

Neural Belief Tracker. For the neural tracker the user input u^t is converted to a vector representation x^t, using a bag of n-grams (with $n = 2$) representation. Each element of x^t is an integer indicating the count of a particular n-gram in u^t. We let $\mathcal{V}^n$ denote the number of unique n-grams, hence $x^t \in \mathbb{N}_0^{V^n}$.

Recurrent neural networks have been used for belief tracking [93, 225] since the output distribution at turn t depends on all user inputs till that turn. We use a GRU (§ 2.2.1) for each tracker, which, starting from $h_j^0 = 0$ computes $h_j^t = \overrightarrow{\text{GRU}}(x^1, \ldots, x^t)$. $h_j^t \in \mathbb{R}^d$ can be interpreted as a summary of what the user has said about slot j till turn t. The belief states are computed from this vector as follows:

$$\begin{aligned} p_j^t &= \text{softmax}(W_j^p h_j^t + c_j^p) \\ q_j^t &= \sigma(W_j^\Phi h_j^t + c_j^\Phi) \end{aligned} \tag{6.8}$$

Here $W_j^p \in \mathbb{R}^{\mathcal{V}^j \times d}$, $c_j^p \in \mathbb{R}^{\mathcal{V}^j}$, $W_j^\Phi \in \mathbb{R}^d$ and $c_j^\Phi \in \mathbb{R}$, are trainable parameters.

Soft-KB Lookup + Summary

This module uses the Soft-KB lookup described in section 6.2 to compute the posterior $p_\mathcal{T}^t \in \mathbb{R}^N$ over the EC-KB from the belief states (p_j^t, q_j^t). Collectively, outputs of the belief trackers and the soft-KB lookup can be viewed as the current dialogue state internal to the KB-InfoBot. Let $s^t = [p_1^t, p_2^t, ..., p_M^t, q_1^t, q_2^t, ..., q_M^t, p_\mathcal{T}^t]$ be the vector of size $\sum_j \mathcal{V}^j + M + N$ denoting this state. It is possible for the agent to directly use this state vector to select its next action a^t. However, the large size of the state vector would lead to a large number of parameters in the policy network. To improve efficiency we extract summary statistics from the belief states, similar to Williams and Young [231].

Each slot is summarized into an entropy statistic over a distribution w_j^t computed from elements of the KB posterior $p_\mathcal{T}^t$ as follows:

$$w_j^t(v) \propto \sum_{i:\mathcal{T}_{i,j}=v} p_\mathcal{T}^t(i) + p_j^0(v) \sum_{i:\mathcal{T}_{i,j}=\Psi} p_\mathcal{T}^t(i)\,. \tag{6.9}$$

Here, p_j^0 is a prior distribution over the values of slot j, estimated using counts of each value in the KB. The probability mass of v in this distribution is the agent's confidence that the user goal has value v in slot j. The two terms in Eq. 6.9 correspond to rows in KB which have value v, and rows whose value is unknown (weighted by the prior probability that an unknown might

be v). Then the summary statistic for slot j is the entropy $H(w_j^t)$. The KB posterior $p_{\mathcal{T}}^t$ is also summarized into an entropy statistic $H(p_{\mathcal{T}}^t)$.

The scalar probabilities q_j^t are passed as is to the dialogue policy, and the final summary vector is $\tilde{s}^t = [H(\tilde{p}_1^t), ..., H(\tilde{p}_M^t), q_1^t, ..., q_M^t, H(p_{\mathcal{T}}^t)]$. Note that this vector has size $2M + 1$.

Dialogue Policy

The dialogue policy's job is to select the next action based on the current summary state $\tilde{s}^t$ and the dialogue history. We present a hand-crafted baseline and a neural policy network.

Hand-Crafted Policy. The rule based policy is adapted from Wu et al. [236]. It asks for the slot $\hat{j} = \arg\min_j H(\tilde{p}_j^t)$ with the minimum entropy, except if – (i) the KB posterior entropy $H(p_{\mathcal{T}}^t) < \alpha_R$, (ii) $H(\tilde{p}_j^t) < \min(\alpha_T, \beta H(\tilde{p}_j^0)$, (iii) slot j has already been requested Q times. α_R, α_T, β, Q are tuned to maximize reward against the simulator.

Neural Policy Network. For the neural approach, similar to Williams and Zweig [232] and Zhao and Eskenazi [261], we use an RNN to allow the network to maintain an internal state of dialogue history. Specifically, we use a GRU unit followed by a fully-connected layer and softmax nonlinearity to model the policy π over actions in $\mathcal{A}$ ($W^\pi \in \mathrm{R}^{|\mathcal{A}| \times d}$, $c^\pi \in \mathrm{R}^{|\mathcal{A}|}$):

$$h_\pi^t = \overrightarrow{\mathrm{GRU}}(\tilde{s}^1, ..., \tilde{s}^t) \tag{6.10}$$

$$\pi = \text{softmax}(W^\pi h_\pi^t + c^\pi)\,. \tag{6.11}$$

During training, the agent samples its actions from the policy to encourage exploration. If this action is *inform()*, it must also provide an ordered set of entities indexed by $I = (i_1, i_2, \ldots, i_R)$ in the KB to the user. This is done by sampling R items from the KB-posterior $p_{\mathcal{T}}^t$. This mimics a search engine type setting, where R may be the number of results on the first page.

6.4 End-to-End Training

Parameters of the neural components (denoted by θ) are trained using the REINFORCE algorithm [233]. We assume that the learner has access to a reward signal r_t throughout the course of the dialogue, details of which are in the next section. We can write the expected discounted return of the agent under policy π as follows:

$$J(\theta) = \mathbb{E}\left[\sum_{t=0}^{H} \gamma^t r_t\right]. \tag{6.12}$$

Here, the expectation is over all possible trajectories τ of the dialogue, θ denotes the trainable parameters of the learner, H is the maximum length of an episode, and γ is the discounting factor. We also use a baseline reward signal b, which is the average of all rewards in a batch, to reduce the variance in the updates [83]. We can use the likelihood ratio trick [80] to write the gradient of the objective as follows:

$$\nabla_\theta J(\theta) = \mathbb{E}\left[\nabla_\theta \log p_\theta(\tau) \sum_{t=0}^{H} \gamma^t (r_t - b)\right], \tag{6.13}$$

where $p_\theta(\tau)$ is the probability of observing a particular trajectory under the current policy. With a Markovian assumption, we can write

$$p_\theta(\tau) = p(s_0) \prod_{k=0}^{H} p(s_{k+1}|s_k, a_k)\pi_\theta(a_k|s_k), \tag{6.14}$$

where θ denotes dependence on the neural network parameters. From 6.13,6.14 we obtain

$$\nabla_\theta J(\theta) = \mathbb{E}_{a\sim\pi}\Big[\sum_{k=0}^{H} \nabla_\theta \log \pi_\theta(a_k) \sum_{t=0}^{H} \gamma^t (r_t - b)\Big], \tag{6.15}$$

If we need to train both the policy network and the belief trackers using the reinforcement signal, we can view the KB posterior $p_{\mathcal{T}}^t$ as another policy. During training then, to encourage exploration, when the agent selects the *inform* action we sample R results $I = (i_1, i_2, \ldots, i_R)$ from the following distribution to return to the user:

$$\mu(I) = p_{\mathcal{T}}^t(i_1) \times \frac{p_{\mathcal{T}}^t(i_2)}{1 - p_{\mathcal{T}}^t(i_1)} \times \cdots . \tag{6.16}$$

This formulation also leads to a modified version of the episodic REINFORCE update rule [233]. Specifically, Eq. 6.14 now becomes,

$$p_\theta(\tau) = \left[p(s_0) \prod_{k=0}^{H} p(s_{k+1}|s_k, a_k)\pi_\theta(a_k|s_k)\right] \mu_\theta(I), \tag{6.17}$$

Notice the last term μ_θ above, which is the posterior of a set of results from the KB. From Eqs. 6.13 and 6.17 we obtain

$$\nabla_\theta J(\theta) = \mathbb{E}_{a\sim\pi, I\sim\mu}\Big[\big(\nabla_\theta \log \mu_\theta(I) + \sum_{k=0}^{H} \nabla_\theta \log \pi_\theta(a_k)\big) \sum_{t=0}^{H} \gamma^t (r_t - b)\Big]. \tag{6.18}$$

Eq. 6.18 gives us the update rule for optimizing the objective in Eq. 6.12, but in practice we found that, in the case of end-to-end learning, the agent almost always fails if starting from

random initialization. In this case, credit assignment is difficult for the agent, since it does not know whether the failure is due to an incorrect sequence of actions or incorrect set of results from the KB. Hence, at the beginning of training we have an *Imitation Learning* (IL) phase where the belief trackers and policy network are trained to mimic the hand-crafted agents. Assume that $\hat{p}_j^t$ and $\hat{q}_j^t$ are the belief states from a rule-based agent, and $\hat{a}^t$ its action at turn t. Then the loss function for imitation learning is:

$$\mathcal{L}(\theta) = \mathbb{E}\left[\sum_j D(\hat{p}_j^t || p_j^t(\theta)) + H(\hat{q}_j^t, q_j^t(\theta)) - \log \pi_\theta(\hat{a}^t)\right]$$

$D(p||q)$ and $H(p, q)$ denote the KL divergence and cross-entropy between p and q respectively. The expectations are estimated using a mini-batch of dialogues of size B. For optimizing the loss, in the IL phase we use vanilla SGD updates [96] and in the RL phase we use RMSProp [96].

6.5 Experiments & Results

Previous work in KB-based QA has focused on single-turn interactions and is not directly comparable to the present study. Instead we compare different versions of the KB-InfoBot described above to test our claims.

6.5.1 Models & Data

KB-InfoBot versions

We have described two belief trackers, (A) Hand-Crafted and (B) Neural, and two dialogue policies, (C) Hand-Crafted and (D) Neural.

Rule agents use the hand-crafted belief trackers and hand-crafted policy (A+C). **RL** agents use the hand-crafted belief trackers and the neural policy (A+D). We compare three variants of both sets of agents, which differ only in the inputs to the dialogue policy. The *No-KB* version only takes entropy $H(\hat{p}_j^t)$ of each of the slot distributions. The *Hard-KB* version performs a hard-KB lookup and selects the next action based on the entropy of the slots over retrieved results. This is the same approach as in Wen et al. [225], except that we take entropy instead of summing probabilities. The *Soft-KB* version takes summary statistics of the slots and the KB posterior described in § 6.3. At the end of the dialogue, all versions inform the user with the top results from the KB posterior $p_{\mathcal{T}}^t$, hence the difference only lies in the policy for action selection. Lastly, the **E2E** agent uses the neural belief tracker and the neural policy (B+D), with

a Soft-KB lookup. For the RL agents, we also append $\hat{q}_j^t$ and a one-hot encoding of the previous agent action to the policy network input.

We use GRU hidden state size of $d = 50$ for the RL agents and $d = 100$ for the E2E, a learning rate of 0.05 for the imitation learning phase and 0.005 for the reinforcement learning phase, and minibatch size 128. For the rule agents, hyperparameters were tuned to maximize the average reward of each agent in simulations. For the E2E agent, imitation learning was performed for 500 updates, after which the agent switched to reinforcement learning. The input vocabulary is constructed from the NLG vocabulary and bigrams in the KB, and its size is 3078.

User Simulator

Training reinforcement learners is challenging because they need an environment to operate in. In the dialogue community it is common to use simulated users for this purpose [4, 38, 174, 175]. In this work we adapt the publicly-available user simulator presented in Li et al. [126] to follow a simple agenda while interacting with the KB-InfoBot.

At the beginning of each dialogue, the simulated user randomly samples a target entity from the EC-KB and a random combination of *informable slots* for which it knows the value of the target. The remaining slot-values are unknown to the user. The user initiates the dialogue by providing a subset of its informable slots to the agent and requesting for an entity which matches them. In subsequent turns, if the agent requests for the value of a slot, the user complies by providing it or informs the agent that it does not know that value. If the agent informs results from the KB, the simulator checks whether the target is among them and provides the reward.

We convert dialogue acts from the user into natural language utterances using a separately trained natural language generator (NLG). The NLG is trained in a sequence-to-sequence fashion, using conversations between humans collected by crowd-sourcing. It takes the dialogue actions (DAs) as input, and generates template-like sentences with slot placeholders via an LSTM decoder. Then, a post-processing scan is performed to replace the slot placeholders with their actual values, which is similar to the decoder module in Wen et al. [224, 226]. In the LSTM decoder, we apply beam search, which iteratively considers the top k best sentences up to time step t when generating the token of the time step $t + 1$. For the sake of the trade-off between the speed and performance, we use the beam size of 3 in the following experiments.

There are several sources of error in user utterances. Any value provided by the user may be corrupted by noise, or substituted completely with an incorrect value of the same type (e.g., "Bill Murray" might become just "Bill" or "Tom Cruise"). The NLG described above is inher-

KB-split	N	M	$\max_j \lvert V^j \rvert$	$\lvert M_j \rvert$
Small	277	6	17	20%
Medium	428	6	68	20%
Large	857	6	101	20%
X-Large	3523	6	251	20%

Table 6.1: Movies-KB statistics for four splits. Refer to Section 6.2 for description of columns.

ently stochastic, and may sometimes generate utterances irrelevant to the agent request. By increasing the temperature of the output softmax in the NLG we can increase the noise in user utterances.

During training, the simulated user also provides a reward signal at the end of each dialogue. The dialogue is a success if the user target is in top $R = 5$ results returned by the agent; and the reward is computed as $\max(0, 2(1 - (r-1)/R))$, where r is the actual rank of the target. For a failed dialogue the agent receives a reward of -1, and at each turn it receives a reward of -0.1 to encourage short sessions.[4] The maximum length of a dialogue is 10 turns beyond which it is deemed a failure.

Movies-KB

We use a movie-centric KB constructed using the IMDBPy package.[5] We constructed four different splits of the dataset, with increasing number of entities, whose statistics are given in Table 6.1. The original KB was modified to reduce the number of actors and directors in order to make the task more challenging, since otherwise only knowing one slot-value may be sufficient for determining the movie.[6] We randomly remove 20% of the values from the agent's copy of the KB to simulate a scenario where the KB may be incomplete. The user, however, may still know these values.

6.5.2 Simulated User Evaluation

We compare each of the discussed versions along three metrics: the average rewards obtained (R), success rate (S) (where success is defined as providing the user target among top R results),

[4] A turn consists of one user action and one agent action.

[5] `http://imdbpy.sourceforge.net/`

[6] We restricted the vocabulary to the first few unique values of these slots and replaced all other values with a random value from this set.

	Agent	Small KB			Medium KB			Large KB			X-Large KB		
		T	S	R	T	S	R	T	S	R	T	S	R
No KB	Rule	5.04	.64	.26±.02	5.05	.77	.74±.02	4.93	.78	.82±.02	4.84	.66	.43±.02
	RL	2.65	.56	.24±.02	3.32	.76	.87±.02	3.71	.79	.94±.02	3.64	.64	.50±.02
Hard KB	Rule	5.04	.64	.25±.02	3.66	.73	.75±.02	4.27	.75	.78±.02	4.84	.65	.42±.02
	RL	3.36	.62	.35±.02	**3.07**	.75	.86±.02	3.53	.79	.98±.02	**2.88**	.62	.53±.02
Soft KB	Rule	**2.12**	.57	.32±.02	3.94	.76	.83±.02	3.74	.78	.93±.02	4.51	.66	.51±.02
	RL	2.93	.63	.43±.02	3.37	.80	.98±.02	3.79	.83	1.05±.02	3.65	**.68**	**.62±.02**
	E2E	3.13	**.66**	**.48±.02**	3.27	**.83**	**1.10±.02**	**3.51**	**.83**	**1.10±.02**	3.98	.65	.50±.02
Max		*3.44*	*1.0*	*1.64*	*2.96*	*1.0*	*1.78*	*3.26*	*1.0*	*1.73*	*3.97*	*1.0*	*1.37*

Table 6.2: Performance comparison. Average (±std error) for 5000 runs after choosing the best model during training. T: Average number of turns. S: Success rate. R: Average reward.

and the average number of turns per dialogue (T). For the RL and E2E agents, during training we fix the model every 100 updates and run 2000 simulations with greedy action selection to evaluate its performance. Then after training we select the model with the highest average reward and run a further 5000 simulations and report the performance in Table 6.2. For reference we also show the performance of an agent which receives perfect information about the user target without any errors, and selects actions based on the entropy of the slots (*Max*). This can be considered as an upper bound on the performance of any agent [236].

In each case the Soft-KB versions achieve the highest average reward, which is the metric all agents optimize. In general, the trade-off between minimizing average turns and maximizing success rate can be controlled by changing the reward signal. Note that, except the E2E version, all versions share the same belief trackers, but by re-asking values of some slots they can have different posteriors $p_{\mathcal{T}}^t$ to inform the results. This shows that having full information about the current state of beliefs over the KB helps the Soft-KB agent discover better policies. Further, reinforcement learning helps discover better policies than the hand-crafted rule-based agents, and we see a higher reward for RL agents compared to Rule ones. This is due to the noisy natural language inputs; with perfect information the rule-based strategy is optimal. Interestingly, the RL-Hard agent has the minimum number of turns in 2 out of the 4 settings, at the cost of a lower success rate and average reward. This agent does not receive any information about the uncertainty in semantic parsing, and it tends to inform as soon as the number of retrieved results becomes small, even if they are incorrect.

Among the Soft-KB agents, we see that E2E>RL>Rule, except for the X-Large KB. For E2E,

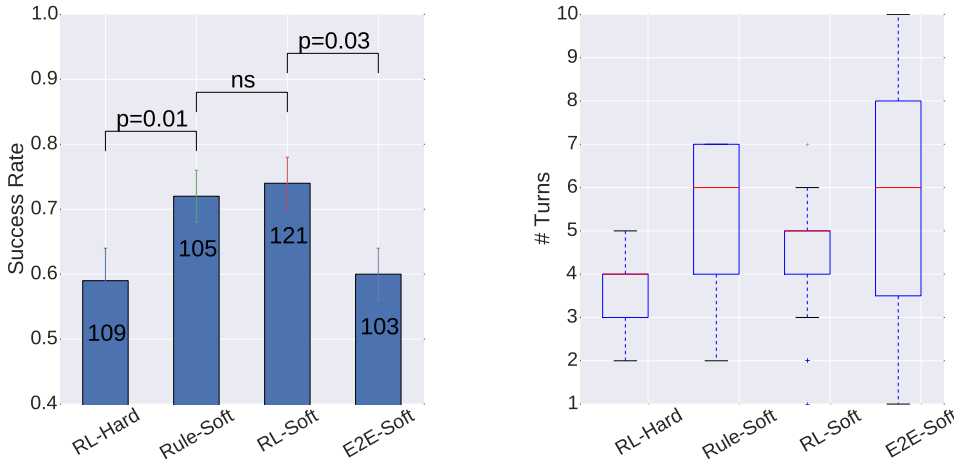


Figure 6.3: Performance of KB-InfoBot versions when tested against humans. (Left) Success rate, with the number of test dialogues indicated on each bar, and the p-values from a two-sided permutation test. (Right) Distribution of the number of turns in each dialogue (differences in mean are significant with $p < 0.01$).

the action space grows exponentially with the size of the KB, and hence credit assignment gets more difficult. The difficulty of a KB-split depends on number of entities it has, as well as the number of unique values for each slot (more unique values make the problem easier). Hence we see that both the "Small" and "X-Large" settings lead to lower reward for the agents, since $\frac{\max_j |\mathcal{V}^j|}{N}$ is small for them.

6.5.3 Human Evaluation

We further evaluate the KB-InfoBot versions trained using the simulator against real subjects.[7] In each session, in a typed interaction, the subject was first presented with a target movie from the "Medium" KB-split along with a subset of its associated slot-values from the KB. To simulate the scenario where end-users may not know slot values correctly, the subjects in our evaluation were presented multiple values for the slots from which they could choose any one while interacting with the agent. Subjects were asked to initiate the conversation by specifying some of these values, and respond to the agent's subsequent requests, all in natural language. We test RL-Hard and the three Soft-KB agents in this study, and in each session one of the agents was picked at random for testing. In total, we collected 433 dialogues, around 20 per subject. Figure 6.3 shows a comparison of these agents and Figure 6.4 shows some sample dialogues

[7]Graduate students at Carnegie Mellon University, and researchers at Microsoft.

Turn	Dialogue	Rank	Dialogue	Rank	Dialogue	Rank
1	can i get a movie directed by maiellaro *request actor*	75	find a movie directed by hemecker *request actor*	7	peter greene acted in a family comedy - what was it? *request actor*	35
2	neal *request mpaa_rating*	2	i dont know *request mpaa_rating*	7	peter *request mpaa_rating*	28
3	not sure about that *request critic_rating*	2	i dont know *request critic_rating*	7	i don't know that *request critic_rating*	28
4	i don't remember *request genre*	2	7.6 *request critic_rating*	13	the critics rated it as 6.5 *inform*	3
5	i think it's a crime movie *inform*	1	7.9 *request critic_rating*	23		
6			7.7 *inform*	41		

Figure 6.4: Sample dialogues between users and the KB-InfoBot (RL-Soft version). Each turn begins with a user utterance followed by the *agent response*. Rank denotes the rank of the target movie in the KB-posterior after each turn.

from RL-Soft.

In comparing Hard-KB versus Soft-KB lookup methods we see that both Rule-Soft and RL-Soft agents achieve a higher success rate than RL-Hard, while E2E-Soft does comparably. They do so in an increased number of average turns, but achieve a higher average reward as well. Between RL-Soft and Rule-Soft agents, the success rate is similar, however the RL agent achieves that rate in a lower number of turns on average. RL-Soft achieves a success rate of 74% on the human evaluation and 80% against the simulated user, indicating minimal overfitting. However, all agents take a higher number of turns against real users as compared to the simulator, due to the noisier inputs.

The E2E gets the highest success rate against the simulator, however, when tested against real users it performs poorly with a lower success rate and a higher number of turns. Since it has more trainable components, this agent is also most prone to overfitting. In particular, the vocabulary of the simulator it is trained against is quite limited ($V^n = 3078$), and hence when real users provided inputs outside this vocabulary, it performed poorly.

6.6 Related Work

This work was motivated by the neural GenQA [252] and neural enquirer [253] models for querying KBs via natural language in a fully "neuralized" way. However, the key difference is that these systems assume that users can compose a complicated, compositional natural language query that can uniquely identify the element/answer in the KB. The research task is to "parse" the query, i.e., turning the natural language query into a sequence of SQL-like opera-

tions. Instead we focus on how to query a KB interactively without composing such complicated queries in the first place. Our work is motivated by the observations that (1) users are more used to issuing simple queries of length less than 5 words [191]; (2) in many cases, it is unreasonable to assume that users can construct compositional queries without prior knowledge of the structure of the KB to be queried.

Dialogue agents can also interface with the database by augmenting their output action space with predefined API calls [19, 127, 232, 261]. The API calls modify a query hypothesis maintained outside the end-to-end system which is used to retrieve results from this KB. This framework does not deal with uncertainty in language understanding since the query hypothesis can only hold one slot-value at a time. Wu et al. [236] presented an entropy minimization dialogue management strategy for InfoBots. The agent always asks for the value of the slot with maximum entropy over the remaining entries in the database, which is optimal in the absence of language understanding errors, and serves as a baseline against our approach. Reinforcement learning neural turing machines (RL-NTM) [257] also allow neural controllers to interact with discrete external interfaces. The interface considered in that work is a one-dimensional memory tape, while in our work it is an entity-centric KB.

6.7 Discussion

This chapter discussed end-to-end trainable dialogue agents for information access. We introduced a differentiable probabilistic framework for querying a database given the agent's beliefs over its fields (or slots). We showed that such a framework allows the downstream reinforcement learner to discover better dialogue policies by providing it more information. We also presented an E2E agent for the task, which demonstrates a strong learning capacity in simulations but suffers from overfitting when tested on real users.

Given these results, we propose the following deployment strategy that allows a dialogue system to be tailored to specific users via learning from agent-user interactions. The system could start off with an RL-Soft agent (which gives good performance out-of-the-box). As the user interacts with this agent, the collected data can be used to train the E2E agent, which has a strong learning capability. Gradually, as more experience is collected, the system can switch from RL-Soft to the personalized E2E agent.

Following our work, Eric et al. [67] proposed a sequence transduction model for training end-to-end task-oriented dialogue agents. Instead of explicitly maintaining beliefs over the user intents, their model directly maps a hidden state computed from the user utterances to a

distribution over the KB contents via a key-value attention mechanism. While this approach lacks the interpretability of modular agents, it benefits from fewer hand-designed components, and may be preferable in data rich settings.

The main limitation of KB-Infobots is scalability, since we compute a posterior distribution over the entire KB, which may be prohibitive for large KBs. In general, this may be addressed by using retrieval techniques similar to those discussed in the previous chapter, so that we can limit tha analysis to a subset of the KB relevant to the user input at the first time-step. However, this may introduce cascading errors—hence in the next two chapters, the last part of this thesis, we explore whether we can directly answer queries against a large knowledge source, without retrieval.

Part III

Text as a Virtual Knowledge Base

Chapter 7

Lazy Slot-Filling

The neural models presented in this thesis so far have all been restricted to processing relatively small contexts composed of text and KB facts, retrieved using shallow methods such as TFIDF. This was dictated largely due to memory constraints imposed by GPUs, the preferred hardware for training these models. This two-step retrieve-plus-read pipeline, however, is not suitable for answering queries which require aggregating information from several passages at once, since all those passages may not be retrieved in the first step. Further, the retrieval step assumes access to a natural language query in the first place, which may not be the case when the knowledge interface is part of a larger system, and the query to it may be implicitly represented using, for example, a vector representation.

In the last part of this thesis, we propose a knowledge representation, called a *Virtual Knowledge Base*, which allows answering queries scalably against an entire text corpus. Virtual KBs combine dense contextual representations of text spans with a sparse index mapping entities to their mentions. In this chapter, we show how to train the dense representations, and answer simple queries against the virtual KB using maximum inner product search (which we term *lazy slot-filling*). In the next chapter, we will show how the additional sparse structure about entities enables fast multi-hop reasoning over the virtual KB.

7.1 Overview

One simple operation on a structured KB is to retrieve triples that match some partial description: for instance from the query HeadquarteredInCity(Microsoft, ?) one might retrieve the "filler" Seattle. These queries are called here *slot-filling queries*. Because large KBs are often incomplete [136], relation extraction methods—e.g., Socher et al. [189]–are often used to

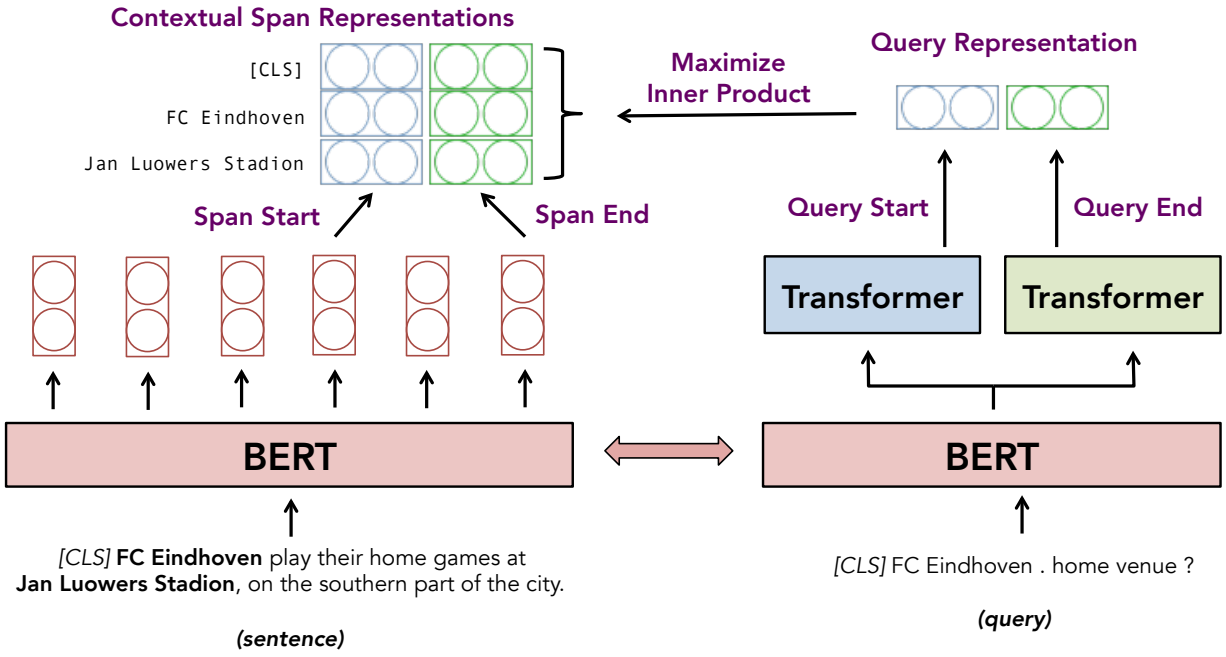


Figure 7.1: The TARDIS model for answering queries by extracting a span from a given sentence. [CLS] is a special token appended to the beginning of each sequence. We train the model to predict [CLS] when the sentence does not contain the answer. The double arrow indicates that the parameters of BERT are shared between the sentence and query. For the probing experiments (§7.3.4), the BERT model (shaded red boxes) parameters are kept fixed.

populate KBs. Here we propose an alternative: instead of performing relation extraction, we *instead directly treat a large corpus as a KB*, and learn to answer slot-filling queries on-the-fly using the corpus. Concretely, a query consists of a *relation* and a *head entity*, provided by the user as text, and the answer to a query is a subspan of the indexed corpus. We call this task *lazy slot-filling* (by analogy to "lazy evaluation" in programming languages) to emphasize that the "slot-filling" queries are answered without a pre-extracted KB .

Our work is motivated by Seo et al. [181], who formulated the task of *phrase-indexed question-answering* (PIQA). In PIQA an arbitrary natural-language question is answered with a span from a corpus, and further, the span must be retrieved using a small number of maximal inner product search (MIPS) queries. A important limitation of PIQA is that it is unclear what sort of information the indexed corpus should store, since any question at all could, in principle, be posed by a user. Here we define a clear goal for the index—it needs to record how entity mentions in the corpus are related. This has similarities with how information is stored in a structured KB, but with the key difference that instead of using discrete edges to represent relations, we use

continuous embeddings computed from text, which can capture arbitrary fine-grained semantics. We also opt for representing *all mentions* in the corpus, which does not lead to any loss of information. By analogy, we term the index a Virtual KB.

Restricting the information in the index naturally restricts the kind of queries it can be used to answer, compared to PIQA. However, we will show that, through the use of pretrained word embeddings, it generalizes to both unseen relations during training and arbitrary natural language queries. The main benefits of this restriction, on the other hand, are easier procedures for sampling negatives during training, and an algorithm for extending to multi-hop queries, which we discuss in the next chapter.

We first present an architecture for learning the contextual representations in the index, called TARDIS (Tokens And Relations in a Dual-encoder Index for Slot-filling, Fig. 7.1). It relies on a dual-encoder [77] system based on BERT [51]: given a slot-filling query, two probe vectors are constructed, one used to retrieve the first token of the answer span, and one to retrieve the last. Using the relation extraction data from Levy et al. [122], we show the applicability of TARDIS representations in the lazy slot-filling setting, where queries are answered against the entire corpus using MIPS. We also present results on relations unseen during training, and propose a novel scheme for describing them by a handful of instances of the slot values they accept (e.g., for "occupation" this could be "scientist" or "politician"). We show that this scheme outperforms the approach of Levy et al. [122], which used question templates to describe a new relation.

7.2 Virtual Knowledge Base

7.2.1 Preliminaries

Let $\mathcal{S} = \{p_i\}_i$ be a corpus, which is a collection of text blocks (sentences in this chapter). Let $\mathcal{K} = \{(s, r, o)\}_j$ be a KB, i.e., a collection of triples describing relation r between the head entity s and the tail entity o. The relations in the seed KB come from a fixed set $\mathcal{R}$, and have string names (e.g., "educated at", "member of sports team"). A slot-filling query is of the form $(s, r, ?)$ where the task is to find the missing tail entity in the triple. Our goal is to answer such queries by extracting spans from $\mathcal{S}$. We are interested in queries which ask about a seen relation $r \in \mathcal{R}$, but a head entity unseen in $\mathcal{K}$ (i.e., in generalizing to unseen entities) as well as queries about an unseen relation $r \notin \mathcal{R}$ and unseen head entity (the "zero-shot" setting).

We answer slot-filling queries through a dual encoder model (§ 7.2.2) that encodes the

answer-containing sentence and query separately, followed by an inner product between the two. The model is trained to extract the answer spans from sentences which contain them (positives), and predict `No-Answer` on those which do not (negatives) (§ 7.2.3). At test time, all spans in the corpus are encoded sentence-by-sentence into their contextual representations, and stored in the virtual KB offline. Queries are answered against these representations by encoding them separately and running MIPS (§ 7.2.3).

7.2.2 Dual Encoder

Figure 7.1 shows an overview of the Dual Encoder that builds on top of the BERT model [51]. We use BERT-base, which is a 12-layer Transformer network, pretrained on a large unsupervised text corpus, which produces contextual representations of an input sequence of tokens. Given a sentence p, we prepend a special token `[CLS]` to it and split it using using WordPiece tokenization [237] to produce a sequence $(w_0, w_1, \ldots, w_N)$. This is encoded through BERT as follows:

$$H_p = [h_0, \ldots, h_N] = \text{BERT}(w_0, \ldots, w_N), \tag{7.1}$$

where $h_i \in \mathbb{R}^d$, and h_0 corresponds to the representation of the special `[CLS]` token. We refer the reader to Devlin et al. [51] for more details.

We consider all spans in the sentence up to a maximum length L as possible answers to a slot-filling query. For a span (i, j) which starts at token i and ends at token j, we take its representation as:

$$H_p^{(i,j)} = [H_p(i); H_p(j)], \tag{7.2}$$

i.e. a concatenation of the BERT outputs at token i and token j. This simple scheme allows efficient inference since it decomposes into two separate inner product searches, as described below.

For the query, we use a string representation q which concatenates the name of the head entity and the name of the relation (e.g. "Microsoft, headquarters?"). This is passed through the same BERT-base model as the sentence to obtain the contextual representations H_q. We further process these query representations into a *start* (q_{st}) and an *end* (q_{en}) vector, respectively, by passing them through two separate 2-layer Transformer networks:

$$\begin{aligned} H_{st} &= \text{Transformer}_{st}(H_q), \\ q_{st} &= H_{st}[0], \end{aligned} \tag{7.3}$$

Query	*(Neil Herron, occupation, rugby union player)*
Positive Instance	Neil Herron is a Scottish rugby union player at the Centre position.
Shared-entity Negative	Neil Herron played for West of Scotland and Glasgow Hawks.
Shared-relation Negative	William Paston, 2nd Earl of Yarmouth was a British peer and politician.

Table 7.1: A slot-filling query with a distantly supervised positive sentence and a shared-entity and shared-relation negative each.

and analogously for q_{en}. The architecture of the Transformer network is identical to that of BERT, but it is randomly initialized. After the final layer, we take the output vector corresponding to position 0 (i.e., the `[CLS]` token). Similar to span representations, the final query representation is a concatenation of the start and end vectors.

Given these representations, we model the probability of a span (i, j) being the answer as:

$$\Pr(a = (i, j)) \propto \exp\left([q_{st}; q_{en}]^T H_p^{(i,j)}\right), \tag{7.4}$$

i.e. an inner-product between the query and span representations. Ideally, the normalization would be over all spans in the sentence, however this is $O(N^2)$ in sentence length. Hence, we adopt a common practice in extractive QA [46, 179] and, during training, normalize the start and end probabilities separately.

7.2.3 Training & Inference

To support inference over a large corpus, we also train over *negative* instances, i.e. query-sentence pairs where the sentence does not contain the answer. This is necessary to drive down the inner product score (Eq. 7.4) between the query and spans in irrelevant sentences in the corpus. For these instances, we use the `[CLS]` span to mean `No-Answer`, and train the model to predict this for negative instances.

The choice of negatives is crucial for avoiding false positives at inference time. The structured nature of the slot-filling task allows us to create two types of targeted negative instances for a given query $(s, r, ?)$, as shown in Table 7.1. In *shared-entity negatives (E-Neg)* the query is paired with a sentence mentioning the subject entity s, but in a different relation r' with a different tail entity o'. In *shared-relation negatives (R-Neg)* the sentence mentions different head and tail entities s' and o' but the same relation r. Levy et al. [122] only used shared-entity negatives for training relation extraction systems. Here, we will show that for lazy slot-filling, both kinds of negatives are important.

Given a dataset of positive as well as negative instances, we train the dual encoder model by minimizing the negative log likelihood of the correct span in Eq. 7.4. After training we index every sentence $p \in \mathcal{S}$ in the corpus using the fine-tuned BERT (Eq. 7.1). The total size of the index is $O(|\mathcal{S}|Nd)$.

At test time, answering queries against the corpus $\mathcal{S}$ leads to the following search problem:

$$\hat{a} = \underset{\substack{p \in \mathcal{S}, i \neq 0, j \neq 0 \\ j > i, j - i < L}}{\arg\max} \; q_{st}^T H_p(i) + q_{en}^T H_p(j) \tag{7.5}$$

In this search we exclude `[CLS]` tokens, so at test time the model never returns `No-Answer`. We perform the search in two steps similar to Seo et al. [182]. First we use only q_{st} to maximize the first term in Eq. 7.5, retaining the top 500 sentence and token index pairs. For this we use FAISS [103], an efficient similarity search library which scales to millions of sentences. Then we maximize Eq. 7.5 only for the spans within these top 500 sentences. In preliminary experiments, we found that this two-stage search results in a negligible drop in accuracy and is much more efficient than enumerating representations of all possible spans.

7.2.4 Generalizing to Unseen Relations

In the zero-shot setting, the query is a relation $r' \notin \mathcal{R}$ not seen in the training data. Because relations are specified in text, some zero-shot generalization does occur, due to the use of pre-trained contextual representations. In preliminary analysis, we found zero-shot generalization was best for unseen relations whose tail entities o have the same *type* as some of the training relations. For example, if the *screenwriter* relation was seen in training, the system might generalize to the *director* relation, but the generalization was poor for unseen relations like *gender* and *occupation*, which have unique types for their tail entities.

Levy et al. [122] reformulated zero-shot slot-filling as a reading comprehension task, by assuming that unseen relations were described by question templates (e.g., "What does *x* do for a living?"). Questions, however, provide only coarse-grained type information. In this work we propose a different zero-shot setting: for a new relation r, the user specifies n *tail type seeds*, $(m_1^r, \ldots, m_n^r)$, of the same type as the expected answer. (E.g., for *occupation* these might be "politician", "scientist", and "soccer player".) We show that for even $n = 3$ such seeds can provide much larger gains in the zero-shot setting than question templates.

To use the tail type seeds, we leverage the insight that BERT's contextual representations cluster entities of the same fine-grained type together (see § 7.3.4). Let M^r denote the set of all mentions of any m_k^r ($k = 1 \ldots n$) in the corpus, each of which is a tuple (p, i, j), detected using

an exact string match. We ensure that M^r contains an equal number of mentions of each of the n instances. We define the tail type representations for r as:

$$q^r_{st} = \frac{1}{|M^r|} \sum_{(p,i,j) \in M^r} H_p(i), \tag{7.6}$$

$$q^r_{en} = \frac{1}{|M^r|} \sum_{(p,i,j) \in M^r} H_p(j), \tag{7.7}$$

When tail type seeds are provided we replace the query in Eq. 7.5 with q'_{st} and q'_{en} below:[1]

$$\begin{aligned} q'_{st} &= q_{st} + q^r_{st}, \\ q'_{en} &= q_{en} + q^r_{en}. \end{aligned} \tag{7.8}$$

Note that the tail type seeds are only used during inference, so training remains unchanged. The idea of using a few sample instances to specify a type, as we do in the tail type instances representation for a relation, is closely related to the task of set expansion [76, 215]. Set expansion has been previously used for a number of tasks, including answering list questions [216], but not zero-shot slot-filling questions.

7.3 Experiments & Results

7.3.1 Setup

Data. We use the slot-filling data released by Levy et al. [122], which consists of triples (s, r, o) from WikiData [212] distantly aligned with sentences in the Wikipedia article of the head entity s which mention both s and o. There are 120 different relations in the data. The data consists of shared-entity negatives (E-Neg, § 7.2.3); we further augment it with shared-relation negatives (R-Neg) by randomly pairing triples with sentences from the dataset aligned to other triples with the same relation. Overall, we end up with 1M training, 1000 validation, and 10K test instances, out of which half are positive, and a quarter each are shared-relation and shared-entity negatives.

Experimental Settings. We begin by studying generalization to unseen entities (§ 7.3.2) and unseen relations (§ 7.3.3) following the methods of Levy et al. [122]. For the latter, we do ten-fold

[1]We experimented with different weights for combining the two query representations, but found an equal-weighted sum to work best across relations.

cross validation, each time training on one subset of relations and testing on the remaining ones. We first consider the *restricted* setting: queries are paired with positive and negative sentences, each mentioning the subject entity, and we report precision, recall and F1 scores of relation extraction. We then consider the *lazy slot-filling* setting, where queries must be answered using retrieval from the full corpus, and report the accuracy of answering queries. While our dual encoder setup can scale to the entire Wikipedia, and we present such experiments in the next chapter, here we test it on a smaller corpus by collecting the full Wikipedia articles of the subject entities of each query in the dev and test sets. Depending on the benchmark, this creates a corpus of 200K-400K sentences. For experiments on unseen relations, we randomly select 3 tail type seeds from all the unique tail entities o present in the data. We report results both with and without queries whose answers are among these tail type seeds, for comparison to prior work.

Implementation Details. We use the uncased BERT-Base model[2], with 12 layers, 12 attention heads, and 768-dim hidden state sizes. The 2-layer Transformer models for further encoding the query into start and end representations are also 768-dim with 12 attention heads. Training is done with batches of 32 and a learning rate of 0.00003. Similar to BERT, we use the Adam optimizer. The maximum length of sentences is set to 64, and longer sentences are split into chunks using a sliding window with stride 16.[3] Answer spans are restricted to a maximum length of $L = 10$ tokens (WordPieces). We use $n = 3$ tail type seeds and analyze the effect of this choice in Figure 7.2. During training, we track the performance on the validation set in the restricted setting to do early stopping. The number of shared-entity and shared-relation negatives is always kept equal.

Compared Systems. We compare our TARDIS model to the *BiDAF* reading comprehension model of [179]. We consider two variants of each system: the *NL Relation* variant, where the query consists of the natural language relation name only; and the *Multiple Templates* variant from Levy et al. [122], where the relation is converted to a natural language query during both training and testing using crowd-sourced templates (e.g. *occupation* becomes "What is X's job?"). When evaluating on unseen relations, we also evaluate TARDIS using 3 tail type seeds as in Eq. 7.8. In the lazy slot-filling setting, we focus on the TARDIS models, and additionally study the different choices of negatives during training (*Random-, E-, E+R-Neg*).

[2] `https://github.com/google-research/bert`

[3] Any chunk which does not end up containing the answer is trained to predict `No-Answer`.

Model	Prec	Rec	F1
BiDAF [122]			
NL Relation	0.882	0.910	0.896
Multiple Templates	0.877	0.913	0.894
TARDIS Dual Encoder (this work)			
NL Relation	**0.934**	**0.959**	**0.946**

Table 7.2: Restricted setting: Micro-averaged precision, recall and F1 on a held-out set of queries about unseen entities over unseen documents. All models are trained using only shared-entity negatives (§7.2.3).

Model	Micro Avg	Macro Avg
NL Relation (E-Neg)	0.035	0.044
NL Relation (Random-Neg)	0.890	0.738
NL Relation (E+R-Neg)	**0.891**	**0.847**

Table 7.3: Lazy slot-filling: micro- and macro-averaged (over relation types) accuracies of lazy slot-filling for TARDIS trained on entity negatives (*E-Neg*), random negatives (*Random-Neg*), and both entity and relation negatives (*E+R-Neg*).

7.3.2 Generalization to Unseen Entities

For the first restricted-setting experiment, we train on all relations, and test on new entities and sentences at test time. Table 7.2 shows that the TARDIS models perform best, even without multiple templates. Levy et al. [122] estimated an upper bound of close to 0.9 in this setting, attributing some errors to mismatches in answer spans (e.g. "1982" v. "December, 1982"). We noticed that the dual encoder model rarely made such errors, leading a higher performance than the BiDAF model.

Table 7.3 shows the performance of TARDIS in the more challenging lazy slot-filling setting. Training using the shared-entity negatives in this setting is not sufficient, since the retrieval method quickly learns to find sentences with the correct relation but the wrong tail entity—a situation not possible in the restricted setting—leading to average accuracy below 5%. This is remedied by introducing shared-relation negatives as well.

Model	Prec	Rec	F1
BiDAF [122]			
NL Relation	0.405	0.286	0.334
Multiple Templates	0.436	0.364	0.396
TARDIS Dual Encoder (This Work)			
NL Relation	0.601	0.489	0.538
+ 3 Tail type seeds	0.607	**0.613**	**0.609**
Multiple Templates	0.605	0.510	0.552
+ 3 Tail type seeds	0.606	0.612	**0.609**

Table 7.4: Restricted setting: Micro-averaged precision, recall and F1 on unseen relations and unseen entities. The results are averaged across 10 folds, where in each fold a different subset of relations are held out for testing. The TARDIS models are trained only on shared-entity negatives. *+ Tail type seeds* denotes the zero-shot setup described in §7.2.4.

Above we argued that discarding the constraint that question are slot-filling queries might make constructing an index very hard—since any question is possible, it would be hard for system to guess what information to encode and index. For lazy slot-filling, in contrast, it is clear what should be encoded: the contextual representation for the tokens in every entity mention should include information about the entity type, and the relationships of that mention to other entities in the same document. Unsurprisingly, performance on this task is much higher than on the unconstrained variant, PIQA: the best result reported by Seo et al. [181] for exact match accuracy is only 0.527.

We also compare to simply randomly sampling negatives (one for each positive instance): this works well for relations with many instances but fails for relations with few instances, as seen from the high micro-average (averaging prediction problems) but relatively lower macro-average (averaging accuracy for each relation). This is because random negatives are more likely to contain distractor entities of the types required by common relations (e.g., countries) but unlikely to contain distractor entities for uncommon types (e.g., vessel class, taxon rank).

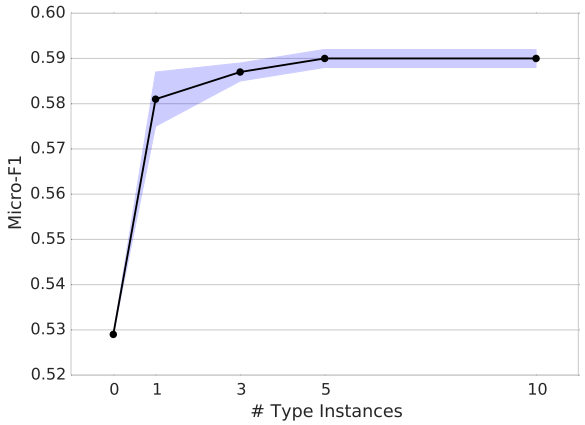


Figure 7.2: Restricted setting: Micro-averaged F1 score on unseen relations, averaged across 10 folds, as the number of tail type seeds provided for each relation increases. For each relation, we repeat the experiment for 3 different randomly selected sets of tail type seeds and show the average and standard deviation (shaded blue region) here.

Model	**Micro Avg**	**Macro Avg**
NL Relation	0.370	0.369
+ 3 Tail type seeds	0.399	0.398
Multiple Templates	0.376	0.372
+ 3 Tail type seeds	**0.411**	**0.406**

Table 7.5: Lazy slot-filling: Micro- and Macro-averaged accuracies on unseen relations, using the TARDIS model. All models are trained on both negative types (*E+R-Neg*). We exclude any query whose answer is among the tail type seeds from the evaluation.

7.3.3 Generalization to Unseen Relations

For the second restricted-setting experiment, we train on ten folds, each time holding out 24 relations (out of 120) for testing, 12 for validation, and the use the rest for training. Table 7.4 shows that the TARDIS models improve substantially on prior work in this zero-shot setting: for example, the NL Relation variant of TARDIS improves the F1 score of BiDAF from 0.334 to 0.538. The type-instance specification of a relation additionally improves performance to 0.609—higher than the performance with multiple templates, which requires crowdsourcing.

Table 7.5 shows the performance of TARDIS in the lazy-slot filling setting on unseen rela-

BERT-Model	Prec	Rec	F1
BERT-Base			
NL Relation (E-Neg)	0.844	0.856	0.850
NL Relation (R-Neg)	0.692	0.733	0.715
Fine-tuned BERT (as in TARDIS)			
NL Relation (E-Neg)	0.866	0.887	0.876
NL Relation (R-Neg)	0.829	0.864	0.846

Table 7.6: Probing experiments: Micro-averaged precision, recall and F1 for pre-trained BERT (uncased BERT-Base) vs Fine-tuned BERT (trained with the TARDIS model using entity and relation negatives *E+R-Neg*).

tions, trained with both shared-entity and shared-relation negatives. On this more challenging task, the addition of the tail type seeds improves macro-averaged accuracy by about three points, over either the relation name alone or the multiple-template description used in prior work.

7.3.4 Further Analysis

Figure 7.2 shows that much of the gain from tail type seeds is obtained from a single instance, and that performance is not improved much by providing more than 3 tail type seeds.[4] Figure 7.2 also shows that the standard deviation of the micro-F1 is quite low as the underlying set of tail type seeds is varied. It is slightly higher when only 1 tail type seed is used.

Table 7.7 gives a number of examples of answers for the same question before and after the addition of tail type seeds. As we would expect, the tail type seeds primarily help in identifying the exact answer span, in some cases via explicit cues, such as "navy" or "cemetery", and in others implicitly by entity types, such as programming languages. For some queries the inclusion of tail types also helps retrieve the correct answer-containing sentence. We also note that the overall performance of the best model in the zero-shot unseen-relation case for lazy slot-filling is comparable to the best prior result for relation extraction in the restricted setting of Levy

[4]In the figure we use NL Relation inputs to the dual encoder model, and train and test it on *both* shared-entity and shared-relation negatives, so the accuracies reported are expected to be slightly different from the results in the table.

et al. [122].

Finally, to compare the representations learned by the BERT model fine-tuned on the lazy slot-filling task, we design two probing experiments. In each experiment, we keep the parameters of the BERT model being probed fixed and only train the query-start and query-end Transformer networks (see Figure 7.1). Similar to Tenney et al. [202], we use a weighted average of the layers of BERT here rather than only the top-most layer, where the weights are learned on the probing task.

In the first experiment, we train and test on shared-entity negatives (E-Neg). Good performance here means the BERT model being probed encodes fine-grained *entity-type* information reliably.[5] As shown in Table 7.6, BERT-Base performs well on this task, suggesting it encodes fine-grained types well.

In the second experiment, we train and test only on shared-relation negatives (R-Neg). Good performance here means that the BERT model encodes *entity co-occurrence* information reliably. In this probe task, we see a large performance drop for BERT-Base, suggesting it does not encode entity co-occurrence information well. The good performance of the TARDIS model on both experiments suggests that fine-tuning on the lazy slot filling task primarily helps the contextual representations to also encode entity co-occurrence information, in addition to entity type information.

7.4 Discussion

Just like a structured KB, the principle behind a virtual KB is to preprocess unstructured textual data into a format such that queries can be answered easily and quickly against it. The difference between the two is that one extracts a discrete set of entities and relations from text, while the other embeds all spans in a continuous space of contextual representations. We have shown that the latter approach can generalize to unseen entities and relations at test time, which the former cannot. The key ingredient which allows this generalization is the pretrained BERT language model. A similar result was shown by Levy et al. [122], but using pretrained word embeddings.

We experimented with a relatively small-scale setting of 200K sentences. Seo et al. [182] showed that for larger corpora, dense embeddings alone are not sufficient for accurate retrieval of answers, and extended the index with sparse embeddings. We will also adopt this approach

[5] A reasonable heuristic for solving this task is to simply detect an entity with the correct type in the given sentence, since all sentences contain the subject entity.

in the next chapter to scale up. While our focus was on constructing the virtual KB from text alone, it is conceptually straightforward to extend the idea to a combination of text and KBs, in a setting similar to Chapter 5.

Traditional KBs are useful beyond just slot-filling. In particular, by aggregating information along entity nodes, they allow several forms of compositional and numerical reasoning over their contents. Developing a similar machinery for virtual KBs is the natural next step, and this will be our focus in the next chapter. It will also be interesting to see how virtual KBs can be integrated into models for tasks beyond QA, such as language modeling.

Query & Answer	Tail type seeds	Predictions
Q. Alison Elizabeth Taylor, educated at ? A. columbia university	Ohio State University, College Notre Dame de Jamhour, University of Durham	*NL Relation:* alison elizabeth taylor is a graduate of **columbia university , school of the arts** and has had three solo ...
		+ type seeds: alison elizabeth taylor is a graduate of **columbia university** , school of the arts and has had three solo ...
Q. Martin Waghorn, military branch ? A. royal navy	Luftwaffe, United States Navy, Pakistan Navy	*NL Relation:* michael waddell (22 december 1922 – 22 may 2015) was a british army officer of the **50th royal tank regiment** ...
		+ type seeds: martin waghorn (died 17 december 1787) was an officer of the **royal navy** .
Q. John Hicklin Hall, place of burial ? A. river view cemetery	Arlington Cemetery, Powazki Cemetery, Gate of Heaven Cemetery	*NL Relation:* john hicklin hall died in portland , oregon , and was interred at river view cemetery in **portland** .
		+ type seeds: john hicklin hall died in portland , oregon , and was interred at **river view cemetery** in portland .
Q. AsciiDoc, programming language ? A. python	Go, Mocklisp, OCaml	*NL Relation:* the main language is **spanish** .
		+ type seeds: asciidoc was created in ... written in the **python** programming language to convert plaintext files to ...

Table 7.7: Lazy slot-filling predictions about unseen relations from the dual encoder model. We show the retrieved sentence and answer (in bold) from the NL Relation model trained on E+R-Neg, and the one which additionally uses 3 tail type seeds.

Chapter 8

Differentiable Reasoning

Traditional Knowledge Bases organize information around entities, which makes it easy to reason over their contents. For example, given a query like *"When was the Grateful Dead's lead singer born?"*, one can identify the entity *Grateful Dead* and the path of relations *LeadSinger*, *BirthDate* to efficiently extract the answer—provided that this information is present in the KB. In this chapter, we develop a model for answering such questions against the virtual KB described in the previous chapter. In addition to accuracy of the model, we will also focus on speed and scalability—an important consideration in practical settings. This work first appeared as:

- Bhuwan Dhingra, Manzil Zaheer, Vidhisha Balachandran, Graham Neubig, Ruslan Salakhutdinov, and William W Cohen. Differentiable reasoning over a virtual knowledge base. In *Proceedings of the International Conference on Learning Representations*, 2020

Code to reproduce the experiments in this chapter is open-sourced.[1]

8.1 Overview

Answering complex queries, like the one above, is difficult for existing open-domain QA systems that first retrieve and then encode documents from the corpus in a query-dependent fashion [26, 51]. If the passages stating that *"Jerry Garcia was the lead singer of the Grateful Dead"* and *"Jerry Garcia was born in 1942"* are far apart in the corpus, it is difficult for systems that retrieve and read a single passage to find an answer—even though in this example, it might be easy to answer the question after the relations were explicitly extracted into a KB. More gener-

[1] `http://www.cs.cmu.edu/bdhingra/pages/drkit.html`

ally, complex questions involving sets of entities or paths of relations may require aggregating information from multiple documents, which is expensive.

One step towards efficient QA is the virtual KB approach presented in the previous chapter, and also explored in PIQA [181, 182], where spans in the text corpus are associated with question-independent contextual representations and then indexed for fast retrieval. Natural language questions are then answered by converting them into vectors that are used to perform maximum inner product search (MIPS) against the index. This can be done efficiently using approximate algorithms [187]. However, this approach cannot be directly used to answer complex queries, since by construction, the information stored in the index is about the local context around a span—it can only be used for questions where the answer can be derived by reading a single passage.

Here we address this limitation by extending the virtual KB with a sparse structure connecting a predefined set of entities to their mentions in the text (Figure 8.1). Using this additional structure, we introduce an efficient, end-to-end differentiable framework for doing complex QA over a large text corpus. Specifically, we consider "multi-hop" complex queries which can be answered by repeatedly executing a "soft" version of the operation below, defined over a set of entities X and a relation R:

$$Y = X.\text{follow}(R) = \{x' : \exists x \in X \text{ s.t. } R(x, x') \text{ holds}\} \tag{8.1}$$

In past work soft, differentiable versions of this operation were used to answer multi-hop questions against an explicit KB [36]. Here we propose a more powerful neural module which approximates this operation against an indexed corpus (the virtual KB). In our module, the input X is a sparse-vector representing a weighted set of entities, and the relation R is a dense feature vector, e.g. a vector derived from a neural network over a natural language query. X and R are used to construct a MIPS query used for retrieving the top-K spans from the index. The output Y is another sparse-vector representing the weighted set of entities, aggregated over entity mentions in the top-K spans.

For multi-hop queries, the output entities Y can be recursively passed as input to the next iteration of the same module. The weights of the entities in Y are differentiable w.r.t the MIPS queries, which allows end-to-end learning *without* any intermediate supervision. We discuss an implementation based on sparse-matrix-vector products, whose runtime and memory depend *only* on the number of spans K retrieved from the index. This is crucial for scaling up to large corpora, providing up to 15x faster inference than existing state-of-the-art multi-hop and open-domain QA systems. The system we introduce is called DrKIT (for Differentiable Reasoning

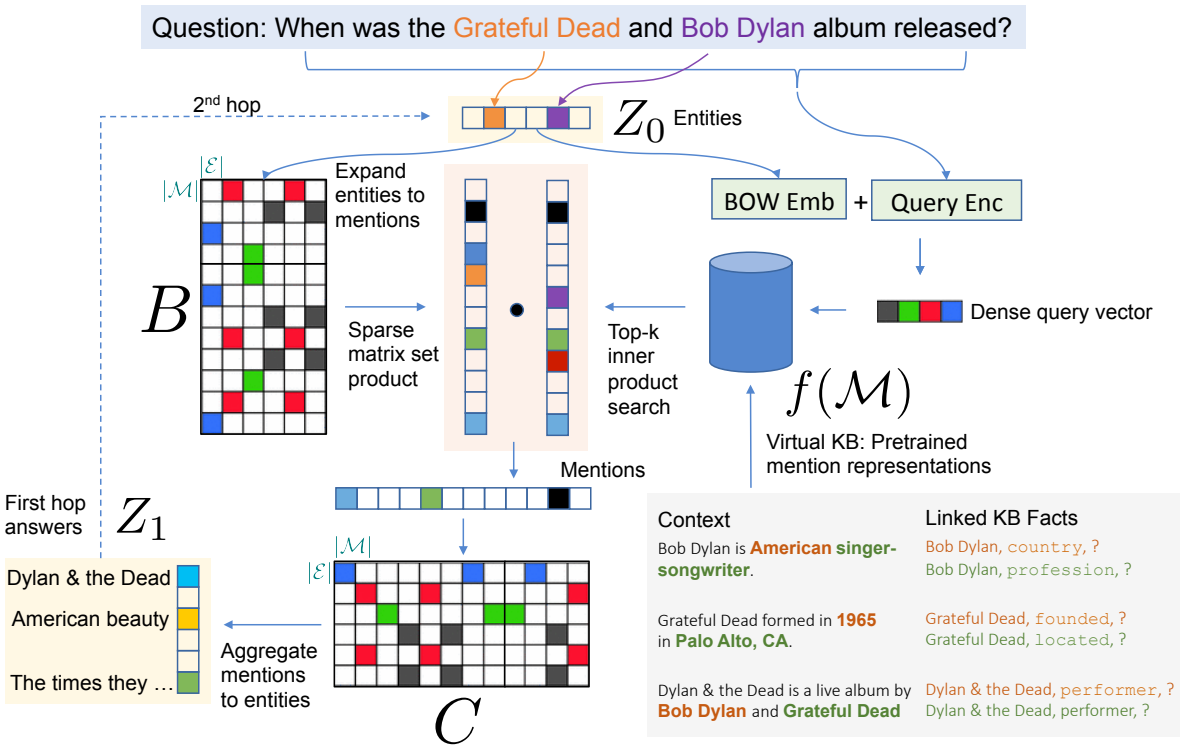


Figure 8.1: DrKIT answers multi-hop questions by iteratively mapping an input set of entities X (*The Grateful Dead*, *Bob Dylan*) to an output set of entities Y (*Dylan & the Dead*, *American beauty*, ...) which are related to any input entity by some relation R (*album by*).

over a Knowledge base of Indexed Text).

We test DrKIT on the MetaQA benchmark for complex question answering, and show that it improves on prior text-based systems by 5 points on 2-hop and 9 points on 3-hop questions, reducing the gap between text-based and KB-based systems by 30% and 70%, respectively. We also test DrKIT on a new dataset of multi-hop slot-filling over Wikipedia articles, and show that it outperforms DrQA [26] and PIQA [182] adapted to this task. Finally, we apply DrKIT to multi-hop information retrieval on the HotpotQA dataset [246], and show that it significantly improves over a BERT-based reranking approach, while being 10x faster.

8.2 Differentiable Reasoning over a KB of Indexed Text

We want to answer a question q using a text corpus as if it were a KB. We start with the set of entities z in the question q, and would ideally want to follow relevant outgoing relation edges in the KB to arrive at the answer. To simulate this behaviour on text, we first expand z to a

set of co-occurring mentions m (say using TFIDF). Not all of these co-occurring mentions are relevant for the question q, so we train a neural network which filters the mentions based on a relevance score of q to m. Then we can aggregate the resulting set of mentions m to the entities they refer to, ending up with an ordered set z' of entities which are answer candidates, very similar to traversing the KB. Furthermore, if the question requires more than one hop to answer, we can repeat the above procedure starting with z'. This is depicted pictorially in Figure 8.1.

We begin by first formalizing this idea in a probabilistic framework in § 8.2.1. In § 8.2.2, we describe how the expansion of entities to mentions and the filtering of mentions can be performed efficiently, using sparse-matrix products and MIPS algorithms [103]. Lastly, we discuss a pretraining scheme based on the previous chapter for constructing a virtual KB of mention representations suited for this task in § 8.2.3.

8.2.1 Differentiable Multi-Hop Reasoning

We denote the given corpus as $\mathcal{S} = \{s_1, s_2, \ldots\}$, where each $s_k = (s_k^1, \ldots, s_k^{N_k})$ is a sequence of tokens. We start by running an entity linker over the corpus to identify mentions of a fixed set of entities $\mathcal{N}$. Each mention m is a tuple (v_m, k_m, i_m, j_m) denoting that the text span $s_{k_m}^{i_m}, \ldots, s_{k_m}^{j_m}$ in document k_m mentions the entity $v_m \in \mathcal{N}$, and the collection of all mentions in the corpus is denoted as $\mathcal{M}$. Note that typically $|\mathcal{M}| \gg |\mathcal{N}|$.

We assume a *weakly supervised* setting where during training we only know the final answer entities $A \in \mathcal{N}$ for a T-hop question. We denote the latent sequence of entities which answer each of the intermediate hops as $z_0, z_1, \ldots, z_T \in \mathcal{N}$, where z_0 is mentioned in the question, and $z_T = A$. We can recursively write the probability of an intermediate answer as:

$$\Pr(z_t|q) = \sum_{z_{t-1} \in \mathcal{E}} \Pr(z_t|q, z_{t-1}) \Pr(z_{t-1}|q). \tag{8.2}$$

Here $\Pr(z_0|q)$ is the output of an entity linking system over the question, and $\Pr(z_t|q, z_{t-1})$ corresponds to a single-hop model which answers the t-th hop, *given* the entity from the previous hop z_{t-1}, by following the appropriate relation. Eq. 8.2 models reasoning over a chain of latent entities, but when answering questions over a text corpus, we must reason over entity *mentions*, rather than entities themselves. Hence $\Pr(z_t|q, z_{t-1})$ needs to be aggregated over all mentions of z_t, which yields

$$\Pr(z_t|q) = \sum_{m \in \mathcal{M}} \sum_{z_{t-1} \in \mathcal{N}} \Pr(z_t|m) \Pr(m|q, z_{t-1}) \Pr(z_{t-1}|q). \tag{8.3}$$

The interesting term to model in the above equation is $\Pr(m|q, z_{t-1})$, which represents the relevance of mention m given the question and entity z_{t-1}. Following the analogy of a KB, we first expand the entity z_{t-1} to co-occuring mentions m and use a learned scoring function to find the relevance of these mentions. Formally, let $F(m)$ denote a TFIDF vector for the document containing m, $G(z_{t-1})$ be the TFIDF vector of the *surface form* of the entity from the previous hop, and $\phi_t(m, z, q)$ be a learnt scoring function (different for each hop). Thus, we model $\Pr(m|q, z_{t-1})$ as

$$\Pr(m|q, z_{t-1}) \propto \underbrace{\mathbb{1}\{G(z_{t-1}) \cdot F(m) > \epsilon\}}_{\text{expansion to co-occurring mentions}} \times \underbrace{\phi_t(m, z_{t-1}, q)}_{\text{relevance filtering}}. \tag{8.4}$$

Another equivalent way to look at our model in Eq. 8.4 is that the second term retrieves mentions of the correct *type* requested by the question in the t-th hop, and the first term filters these based on co-occurrence with z_{t-1}. When dealing with a large set of mentions m, we will typically retain only the top-K relevant mentions. We will show that this joint modelling of co-occurrence and relevance is important for good performance, as was also observed by Seo et al. [182].

The other term left in Eq. 8.3 is $\Pr(z|m)$, which is 1 if mention m refers to the entity z else 0, based on the entity linking system. In general, to compute Eq. 8.3 the mention scoring of Eq. 8.4 needs to be evaluated for all latent entity and mention pairs, which is prohibitively expensive. However, by restricting ϕ_t to be an inner product we can implement this efficiently (§ 8.2.2).

To highlight the differentiability of the proposed overall scheme, we can represent the computation in Eq. 8.3 as matrix operations. We pre-compute the inner products between TFIDF vectors (§ 2.2.3) for all entities and mentions into a *sparse* matrix, which we denote as an $|\mathcal{N}| \times |\mathcal{M}|$ matrix $B[v, m] = \mathbb{1}\left(G(v)^T F(m) > \epsilon\right)$. Then entity expansion to co-occuring mentions can be done using a sparse-matrix by sparse-vector multiplication between B and z_{t-1}.

For the relevance scores, let $\mathbb{T}_K(\phi_t(m, z_{t-1}, q))$ denote the top-K relevant mentions encoded as a *sparse* vector in $\mathbb{R}^{|\mathcal{M}|}$. Finally, the aggregation of mentions to entities can be formulated as multiplication with another $|\mathcal{M}| \times |\mathcal{N}|$ sparse-matrix C, which encodes *coreference*, i.e. $C[m, v] = 1$ if mention m corresponds to the entity n. Putting all these together, using $\odot$ to denote element-wise product, and defining $Z_t = [\Pr(z_t = v_1|q); \ldots; \Pr(z_t = v_{|\mathcal{N}|}|q)]$, we can observe that for large K (i.e., as $K \to |\mathcal{M}|$), Eq. 8.3 becomes equivalent to:

$$Z_t = \text{softmax}\left(\left[Z_{t-1}^T B \odot \mathbb{T}_K(\phi_t(m, z_{t-1}, q))\right] C\right). \tag{8.5}$$

Note that every operation in above equation is *differentiable* and between *sparse* matrices and vectors: we will discuss efficient implementations in § 8.2.2. Further, the number of non-zero

entries in Z_t is bounded by K, since we filtered (the element-wise product in Eq. 8.5) to top-K relevant mentions among TFIDF based expansion and since each mention can only point to a single entity in C. This is important, as it prevents the number of entries in Z_t from exploding across hops (which might happen if, for instance, we added the relevance and TFIDF scores instead).

We can view Z_{t-1}, Z_t as weighted multisets of entities, and $\phi_t(m, z, q)$ as implicitly selecting mentions which correspond to a relation R. Then Eq. 8.5 becomes a differentiable implementation of $Z_t = Z_{t-1}.\text{follow}(R)$, i.e. mimicking the graph traversal in a traditional KB. We thus call Eq. 8.5 a *textual follow operation*.

Training and Inference. The model is trained end-to-end by optimizing the cross-entropy loss between Z_T, the weighted set of entities after T hops, and the ground truth answer set A.[2] We use a temperature coefficient λ when computing the softmax in Eq, 8.5 since the inner product scores of the top-K retrieved mentions are typically high values, which would otherwise result in very peaked distributions of Z_t. We also found that taking a *maximum* over the mentions of an entity in Eq. 8.3 works better than taking a sum. This corresponds to optimizing only over the most confident mention of each entity, which works for corpora like Wikipedia that do not have much redundancy. A similar observation was made by Min et al. [137] in weakly supervised settings.

8.2.2 Efficient Implementation

Sparse TFIDF Mention Encoding. To compute the sparse-matrix B for entity-mention expansion in Eq. 8.5, the TFIDF vectors $F(m)$ and $G(v)$ are constructed over unigrams and bigrams, hashed to a vocabulary of 16M buckets. We ignore the bottom 160 buckets in terms of IDF scores to encourage sparsity. While F computes the vector from the whole passage around mention m, G only uses the surface form of entity v. This corresponds to retrieving all mentions in a document using v as the query. We limit the number of retrieved mentions per entity by retrieving a maximum of P paragraphs per entity.

Efficient Entity-Mention expansion. The expansion from a set of entities to mentions occurring around them can be computed using the sparse-matrix by sparse-vector product $Z_{t-1}^T B$. A simple lower bound for multiplying a sparse $|\mathcal{N}| \times |\mathcal{M}|$ matrix, with maximum μ non-zeros

[2] We optimize against a uniform distribution over the elements of A.

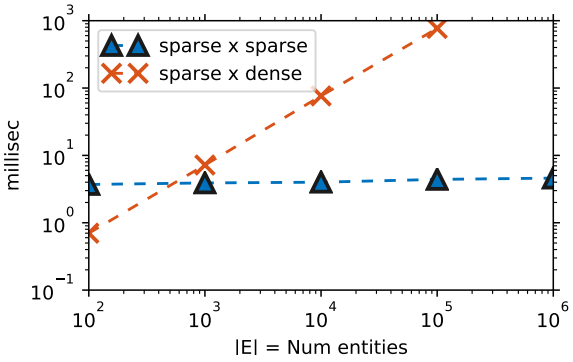


Figure 8.2: Runtime on a single K80 GPU when using ragged representations for implementing sparse-matrix vector product, vs the default sparse-matrix times dense vector product available in TensorFlow. $|\mathcal{N}| > 10^5$ leads to OOM for the latter.

in each row, by a sparse $|\mathcal{N}| \times 1$ vector with K non-zeros is $\Omega(K\mu)$. Note that this lower bound is independent of the size of matrix B, or in other words independent of the number of entities or mentions. To attain the lower bound, the multiplication algorithm must be vector driven, because any matrix-driven algorithms need to at least iterate over all the rows. Instead we *slice* out the relevant rows from B. To enable this our solution is to represent the sparse-matrix B as two row-wise *lists of variable-sized lists* of the indices and values of the non-zero elements, respectively. This results in a "ragged" representation of the matrix [203] which can be easily sliced corresponding to the non-zero entries in the vector in $O(\log |\mathcal{N}|)$ time. We are now left with K sparse-vectors with at most μ non-zero elements in each. We can add these K sparse-vectors weighted by corresponding values from the vector Z_{t-1}^T in $O(K \max\{K, \mu\})$ time. Moreover, such an implementation is feasible with deep learning frameworks such as TensorFlow. We tested the scalability of our approach by varying the number of entities for a fixed density of mentions μ (estimated from Wikipedia). Figure 8.2 compares our approach to the default sparse-matrix times dense-vector product available in TensorFlow.

Efficient top-K mention relevance filtering: To make computation of Eq. 8.5 feasible, we need an efficient way to get top-K relevant mentions related to an entity in z_{t-1} for a given question q, without enumerating all possibilities. To do this, we follow the lazy slot-filling procedure outlined in the last chapter by restricting the scoring function $\phi_t(m, z_{t-1}, q)$ to an inner product. In this manner we can leverage efficient approximate MIPS algorithms (§ 2.2.3). Let $f(m)$ be a dense encoding of m, and $g_t(q, z_{t-1})$ be a dense encoding of the question q for the t-th hop, both in $\mathbb{R}^p$. These are computed in a similar manner to the last chapter (more details

below). Then the scoring function $\phi_t(m, z_{t-1}, q)$ becomes

$$\phi_t(m, z_{t-1}, q) = f(m)^T g_t(q, z_{t-1}), \tag{8.6}$$

We precompute the mention embeddings into a single large matrix $f(\mathcal{M}) = [f(m_1); f(m_2); \ldots]$, and index it for efficient MIPS retrieval [1, 103, 187]. Although this matrix will be very large for a realistic corpus, we can use MIPS to retrieve the top-K values efficiently since we are only interested the top scoring mentions. The complexity of this filtering step using MIPS is roughly $O(Kp \operatorname{polylog}|\mathcal{M}|)$.

Mention and Question Encoders. Mentions are encoded in a similar manner as the last chapter (Eq. 7.2), but using BERT-large instead of BERT-base [51]. To limit the size of the embeddings, and consequently the size of eventual virtual KB, we also linearly project the mention embeddings from BERT. Suppose mention m appears in passage s, starting at position i and ending at position j. Then $f(m) = W^T[H^s_i; H^s_j]$, where H^s is the sequence of embeddings output from BERT, and W is a linear projection to size p.

The queries are also encoded similarly, but with a smaller BERT-like model with only 4 Transformer layers [209], producing the output sequence H^q. Following Eq. 7.3, for each hop $t = 1, \ldots, T$, we add two additional Transformer layers on top of H^q, and take their [CLS] encodings as the start and end vectors, H^q_{st} and H^q_{en}, respectively. We concatenate the two and define $\tilde{g}_t(q) = V^T[H^q_{st}; H^q_{en}]$. Finally, to condition on current progress we add the embeddings of z_{t-1}. Specifically, we use entity embeddings $E \in \mathbb{R}^{|\mathcal{N}| \times p}$, to construct an average embedding of the set Z_{t-1}, as $Z^T_{t-1}E$, and define:

$$g_t(q, z_{t-1}) \equiv \tilde{g}_t(q) + Z^T_{t-1}E. \tag{8.7}$$

To avoid a large number of parameters in the model, we compute the entity embeddings as an average over the word embeddings (first layer in BERT) of the tokens in the entity's surface form. The computational cost of the question encoder $g_t(q)$ is $O(p^2)$.

Complexity. Our total computational complexity to answer a query is $\tilde{O}(K \max\{K, \mu\} + Kp + p^2)$ (almost independent to number of entities or mentions!), with $O(\mu|\mathcal{N}| + p|\mathcal{M}|)$ memory to store the pre-computed matrices and mention index.[3]

[3]Following standard convention, in $\tilde{O}$ notation we suppress poly $\log$ dependence terms.

8.2.3 Pretraining

Ideally, we would like to train the mention encoder $f(m)$ end-to-end using labeled QA data only. However, this poses a challenge when combined with approximate nearest neighbor search, since after every update to the parameters of f, one would need to recompute the embeddings of all mentions in $\mathcal{M}$. We thus adopt a staged training approach: we first pre-train a mention encoder $f(m)$, borrowing ideas from the previous chapter, and then compute and index embeddings for all mentions once, keeping these embeddings fixed when training the downstream QA task. Unlike the lazy slot-filling task, however, note that here the queries to the index (Eq. 8.7) consist of two parts—one derived from a natural language description $\tilde{g}_t(q)$, and the other from the entities in the previous hop $Z_{t-1}^T E$. Hence, we adapt the pretraining procedure accordingly.

Specifically, assume we are given an open-domain KB consisting of facts (e_1, R, e_2) specifying that the relation R holds between the subject e_1 and the object e_2. Then for a corpus of entity-linked text passages $\{s_k\}$, we automatically identify tuples $(s, (e_1, R, e_2))$ such that s mentions both e_1 and e_2. Using this data, we learn to answer slot-filling queries in a reading comprehension setup, where the query q is constructed from the surface form of the subject entity e_1 and a natural language description of R (e.g. "Jerry Garcia. birth place?"), and the answer e_2 needs to be extracted from the passage s. Using string representations in q ensures our pre-training setup is similar to the downstream task. In pretraining, we use the same scoring function as in previous section, but over all *spans* m in the passage:

$$\phi(m, e_1, q) \propto \exp\left\{f(m) \cdot g(q, e_1)\right\}. \tag{8.8}$$

Similar to lazy slot-filling, during training, we normalize the start and end probabilities of picking out the correct span m separately. We use a similar set of negatives as listed in Table 7.1, augmented with some randomly generated ones which pair queries with random text passages from the corpus.

For the multi-hop slot-filling experiments below, we used WikiData [212] as our KB, Wikipedia as the corpus, and SLING [170] to identify entity mentions. We restrict s to be from the Wikipedia article of the subject entity to reduce noise. Overall we collected $950K$ pairs over $550K$ articles. For the experiments with MetaQA, we supplemented this data with the corpus and KB provided with MetaQA, and string matching for entity linking. We use BERT-large in this chapter, which has 24 layers, 16 attention heads in each layer, and an embedding dimension of 1024.[4]

[4] `https://github.com/google-research/bert`.

8.3 Experiments

8.3.1 METAQA: Multi-Hop Question Answering with Text

Dataset. We first evaluate DrKIT on the MetaQA benchmark for multi-hop question answering [260]. MetaQA consists of around $400K$ questions ranging from 1 to 3 hops constructed by sampling relation paths from a movies KB [135] and converting them to natural language using templates. The questions cover 8 relations and their inverses, around $43K$ entities, and are paired with a corpus consisting of $18K$ Wikipedia passages about those entities. The questions are all designed to be answerable using either the KB or the corpus, which makes it possible to compare the performance of our virtual KB QA system to a plausible upper bound system that has access to a complete KB. We used the same version of the data as Sun et al. [197].

Details. We use $p = 400$ dimensional embeddings for the mentions and queries, and 200-dimensional embeddings each for the start and end positions. This results in an index of size 750MB. When computing B, the entity to mention co-occurrence matrix, we only retain mentions in the top $P = 50$ paragraphs matched with an entity, to ensure sparsity. Further we initialize the first 4 layers of the question encoder with the Transformer network from pretraining. For the first hop, we assign Z_0 as a 1-hot vector for the least frequent entity detected in the question using an exact match. The number of nearest neighbors K and the softmax temperature λ were tuned on the dev set of each task, and we found $K = 10000$ and $\lambda = 4$ to work best. We pretrain the index on a combination of the MetaQA corpus, using the KB provided with MetaQA for distance data, and the WikiData corpus.

Results. Table 8.1 shows the accuracy of the top-most retrieved entity (Hits@1) for the subtasks ranging from 1-3 hops, and compares to the state-of-the-art systems for the text-only setting on these tasks. DrKIT outperforms the prior state-of-the-art by a large margin in the 2-hop and 3-hop cases. The strongest prior method, PullNet [196, 197], uses a graph neural network model with learned iterative retrieval from the corpus to answer multi-hop questions. It uses the MetaQA KB during training to identify shortest paths between the question entity and answer entity, which are used to supervise the text retrieval and reading modules. DrKIT, on the other hand, has strong performance without such supervision, demonstrating its capability for end-to-end learning. (Adding the same intermediate supervision to DrKIT does not even consistently improve performance—it gives DrKIT a small lift on 1- and 2-hop questions but does not help for 3-hop questions.)

MetaQA				WikiData			
Model	**1hop**	**2hop**	**3hop**	**Model**	**1hop**	**2hop**	**3hop**
DrQA (ots)	0.553	0.325	0.197	DrQA (ots, cascade)	0.287	0.141	0.070
KVMem†	0.762	0.070	0.195	PIQA (ots, cascade)	0.240	0.118	0.064
GraftNet†	0.825	0.362	0.402	PIQA (pre, cascade)	0.670	0.369	0.182
PullNet†	0.844	0.810	0.782	DrKIT (pre, cascade)	0.816	0.404	0.198
DrKIT (e2e)	0.844	0.860	**0.876**	DrKIT (e2e)	**0.834**	**0.469**	**0.244**
DrKIT (strong sup.)	**0.845**	**0.871**	0.871	–BERT index	0.643	0.294	0.165

Table 8.1: MetaQA (left) and WikiData (right) Hits @1 for 1-3 hop sub-tasks. ots: off-the-shelf without re-training. †: obtained from Sun et al. [197]. cascade: adapted to multi-hop setting by repeatedly applying Eq. 8.3. pre: pre-trained on slot-filling. e2e: end-to-end trained on single-hop and multi-hop queries.

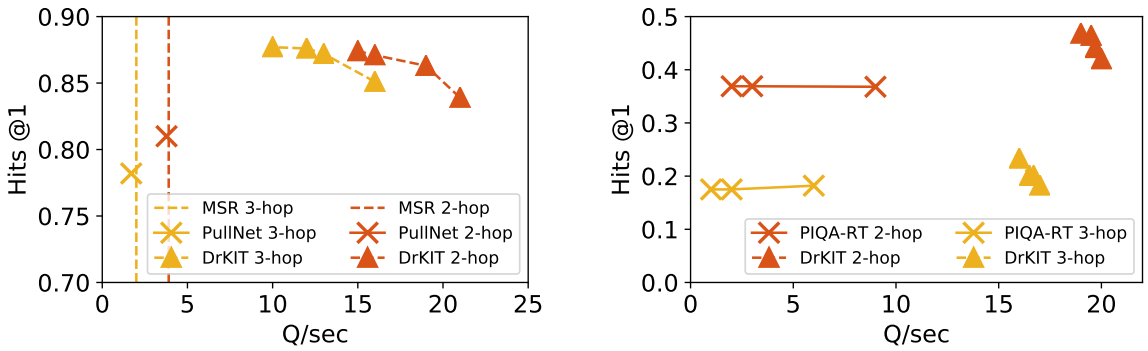


Figure 8.3: Hits @1 vs Queries/sec during inference on MetaQA (left) and WikiData (middle) tasks, measured on a single CPU server with 6 cores. MSR: Multi-step Retriever model from Das et al. [46] (we only show Q/sec).

DrKIT's architecture is driven, in part, by efficiency considerations: unlike PullNet, it is designed to answer questions with minimal processing at query time. Figure 8.3 compares the tradeoffs between accuracy and inference time of DrKIT with PullNet as we vary K, the number of dense nearest neighbors retrieved. The runtime gains of DrKIT over PullNet range between 5x-15x. Figure 8.4 further shows the effect of varying K on the accuracy of the system. We note that K must be somewhere between 5-10K for our approach to be effective.

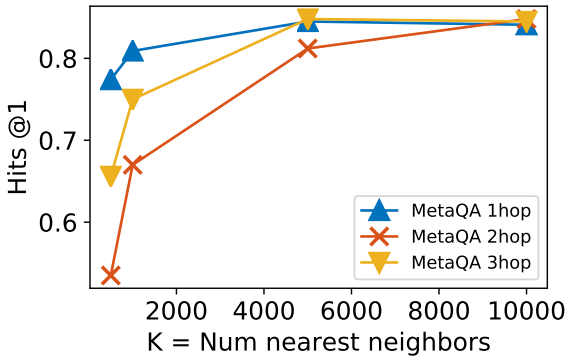


Figure 8.4: Effect of varying number of nearest neighbors K during MIPS on the Hits@1 performance.

Ablations	**1hop**	**2hop**	**3hop**
DrKIT	0.844	0.860	0.876
–Sum over mentions (Eq. 8.3)	0.837	0.823	0.797
$-\lambda = 1$	0.836	0.752	0.799
–w/o TFIDF	0.845	0.548	0.488
–BERT index	0.634	0.610	0.555
Incomplete KB for pretraining			
25% KB	0.839	0.804	0.830
50% KB	0.843	0.834	0.834
(50% KB-only)	*0.680*	*0.521*	*0.597*

Table 8.2: Ablation study for different components of DrKIT.

Analysis. Table 8.2 shows ablations on DrKIT for the MetaQA data. First, we empirically confirm that taking a sum instead of max over the mentions of an entity hurts performance. So does removing the softmax temperature (by setting $\lambda = 1$). Removing the TFIDF component from Eq. 8.4, leads a large decrease in performance for 2-hop and 3-hop questions. This is because the TFIDF component *constrains* the end-to-end learning to be along reasonable paths of co-occurring mentions, preventing the search space from exploding. The results also highlight the importance of the pretraining method of § 8.2.3, as DrKIT over an index of BERT representations without pretraining is 23 points worse in the 3-hop case. We also check the

performance when the KB used for pre-training is *incomplete*. Even with only 50% edges retained, we see good performance—better than PullNet and the state-of-the-art for a KB-only method (in *italics*).

We analyzed 100 2-hop questions correctly answered by DrKIT and found that for 83, the intermediate answers were also correct. The other 17 cases were all where the second hop asked about *genre*, e.g. "What are the genres of the films directed by Justin Simien?". We found that in these cases the intermediate answer was the same as the correct final answer—essentially the model learned to answer the question in 1 hop and copy it over for the second hop. Among incorrectly answered questions, the intermediate accuracy was only 47%, so the mistakes were evenly distributed across the two hops.

8.3.2 WikiData: Multi-Hop Slot-Filling

The MetaQA dataset has been fairly well-studied, but has limitations since it is constructed over a small KB. In this section we consider a new task, in a larger scale setting with many more relations, entities and text passages. The new dataset also lets us evaluate performance in a setting where the test set contains documents and entities not seen at training time, an important issue when devising a QA system that will be used in a real-world setting, where the corpus and entities in the discourse change over time, and lets us perform analyses not possible with MetaQA, such as extrapolating from single-hop to multi-hop settings without retraining.

Task	**#train**	**#dev**	**#test**	$\lvert\mathcal{N}_{test}\rvert$	$\lvert\mathcal{M}_{test}\rvert$	$\lvert\mathcal{S}_{test}\rvert$	**Example**
1hop	16901	2467	10000	216K	1.2M	120K	Q. Mendix, industry? A. Enterprise Software
2hop	163607	398	9897	342K	1.9M	120K	Q. 2000 Hel van het Mergelland, winner, place of birth? A. Bert Grabsch → Lutherstadt Wittenberg
3hop	36061	453	9899	261K	1.8M	120K	Q. Magnificent!, record label, founded by, date of death? A. Prestige → Bob Weinstock → 14 Jan 2006

Table 8.3: WikiData multi-hop slot-filling dataset

Dataset. We sample two subsets of Wikipedia articles, one for pre-training (§ 8.2.3) and end-to-end training, and one for testing. For each subset we consider the set of WikiData entities mentioned in the articles, and sample paths of 1-3 hop relations among them, ensuring that any intermediate entity has an in-degree of no more than 100. Then we construct a semi-structured query by concatenating the surface forms of the head entity with the path of relations (e.g. "Helene Gayle, employer, founded by, ?"). The answer is the tail entity at the end of the path, and the task is to extract it from the Wikipedia articles. In the last chapter, we focused on a *single-hop*, *static* corpus setting, whereas here our task considers a *dynamic* setting which requires the system to traverse the corpus. For each setting, we create a dataset with 10K articles, 120K passages, $>$ 200K entities and 1.5M mentions, resulting in an index of size about 2GB. Details of the collected dataset are shown in Table 8.3.

Baselines. We adapt two publicly available open-domain QA systems for this task – DrQA[5] [26] and PIQA[6] [182]. While DrQA is relatively mature and widely used, PIQA is recent, and similar to our setup, since it also answers questions with minimal computation at query time. PIQA also uses MIPS queries, similar to the lazy slot-filling setup of the last chapter, with an additional sparse component in the index, but cannot be used to answer multi-hop queries. We thus also consider a cascaded architecture which repeatedly applies Eq. 8.3, using either of PIQA or DrQA to compute $\Pr(z_t|q, z_{t-1})$ against the corpus, retaining at most k intermediate answers in each step. We tune k in the range of 1-10, since larger values make the runtime infeasible. Further, since these models were trained on natural language questions, we use the templates released by Levy et al. [122] to convert intermediate questions into natural text.[7] We test off-the-shelf versions of these systems, as well as a version of PIQA re-trained on our our slot-filling data.[8] We compare to a version of DrKIT trained only on single-hop queries (§ 8.2.3) and similarly cascaded, and one version trained end-to-end on the multi-hop queries.

Results. Table 8.1 (right) lists the Hits @1 performance on this task. Off-the-shelf open-domain QA systems perform poorly, showing the challenging nature of the task. Re-training PIQA on the slot-filling data improves performance considerably, but DrKIT trained on the

[5] `https://github.com/facebookresearch/DrQA`

[6] `https://github.com/uwnlp/denspi`

[7] For example, "Helene Gayle. employer?" becomes "Who is the employer of Helene Gayle?"

[8] We tuned several hyperparameters of PIQA on our data, eventually picking the *sparse first* strategy, a sparse weight of 0.1, and a filter threshold of 0.2. For the SQuAD trained version, we also had to remove paragraphs smaller than 50 tokens since with these the model failed completely.

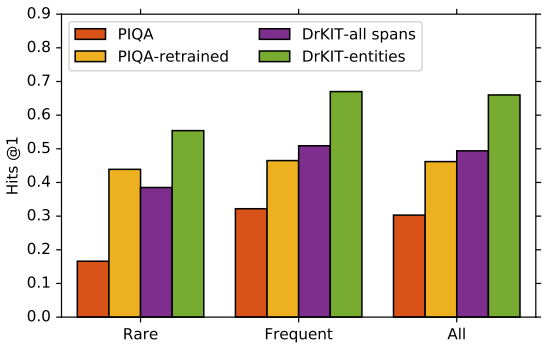


Figure 8.5: Macro-avg accuracy on lazy slot-filling. We split the results based on frequency of the relations in our WikiData training data. DrKIT-all spans refers to a variant of our model which selects from all spans in the corpus, instead of only entity-linked mentions.

same data improves on it. A large improvement over these cascaded architectures is seen with end-to-end training, which is made possible by the differentiable operation introduced in this paper. We also list the performance of DrKIT when trained against an index of fixed BERT-large mention representations. While this is comparable to the re-trained version of PIQA, it lags behind DrKIT pre-trained using the KB, once again highlighting the importance of lazy slot-filling style pretraining. We also plot the Hits @1 against Queries/sec for cascaded versions of PIQA and DrKIT in Figure 8.3 (middle). We observe runtime gains of 2x-3x to DrKIT due to the efficient implementation of entity-mention expansion of §8.2.2.

Analysis. In order to understand where the accuracy gains for DrKIT come from, we conduct further experiments on the lazy slot-filling data constructed in the previous chapter. We report results on 2 subsets of relations in addition to all relations. The *Rare* subset comprises of relations with frequencies < 5 in the training data while the *Frequent* subset contains the rest. PIQA trained on SQuAD only gets 30% macro-avg accuracy on this data, but this improves to 46% when re-trained on our slot-filling data. Interestingly, a version of DrKIT which selects from *all spans* in the corpus performs similarly to PIQA (50%), but when using entity linking it significantly improves to 66%. It also has 55% accuracy in answering queries about rare relations.

Model	Q/s	Accuracy			
		@2	@5	@10	@20
BM25†	–	0.093	0.191	0.259	0.324
PRF-Task†	–	0.097	0.198	0.267	0.330
BERT re-ranker†	–	0.146	0.271	0.347	0.409
Entity Centric IR†	0.32*	0.230	0.482	0.612	0.674
DrKIT (WikiData)		0.355	0.588	0.671	**0.710**
DrKIT (Hotpot)	**4.26***	**0.385**	0.595	0.663	0.703
DrKIT (Combined)		0.383	**0.603**	**0.672**	**0.710**

Model	EM	F1
Baseline†	0.288	0.381
+EC IR‡	0.354	0.462
+Golden Ret◇	**0.379**	**0.486**
+DrKIT†	0.357	0.466

Table 8.4: Left: Retrieval performance on the HotpotQA benchmark dev set. Q/s denotes the number of queries per second during inference on a single 16-core CPU. Accuracy @k is the fraction where *both* the correct passages are retrieved in the top k. †: Baselines obtained from Das et al. [47]. For DrKIT, we report the performance when the index is pretrained using the WikiData KB alone, the HotpotQA training questions alone, or using both. *: Measured on different machines with similar specs. Right: Overall performance on the HotpotQA task, when passing 10 retrieved passages to a downstream reading comprehension model [246]. ‡: From Das et al. [47]. ◇: From Qi et al. [159]. †: Results on the dev set.

8.3.3 HotpotQA: Multi-Hop Information Retrieval

Dataset. HotpotQA [246] is a recent dataset of over 100K crowd-sourced multi-hop questions and answers over introductory Wikipedia passages. We focus on the open-domain *fullwiki* setting where the two gold passages required to answer the question are not known in advance. The answers are free-form spans of text in the passages, not necessarily entities, and hence our model which selects entities is not directly applicable here. Instead, inspired by recent works [47, 159], we look at the challenging sub-task of *retrieving* the passages required to answer the questions, from a pool of 5.23M. This is a multi-hop IR task, since for many questions at least one passage may be 1-2 hops away from the entities in the question. Further, each passage is about an entity (the title entity of that Wikipedia page), and hence retrieving passages is the same as identifying the title entities of those passages. We apply DrKIT to this task of identifying the two entities for each question, whose passages contain the information needed to answer that question. Then we pass the top 10 passages identified this way to a standard reading comprehension architecture from Yang et al. [246] to select the answer span.

Setup. We use the Wikipedia abstracts released by Yang et al. [246] as the text corpus.[9] The total number of entities is the same as the number of abstracts, 5.23M, and we consider hyperlinks in the text as mentions of the entities to whose pages they point to, leading to 22.8M total mentions in an index of size 34GB. For pretraining the mention representations, we compare using the WikiData KB as described in § 8.2.3 to directly using the HotpotQA training questions, with TFIDF based retrieved passages as negative examples. We set $B[v, m] = 1$ if either the entity v is mentioned on the page of the entity denoted by m, or vice versa. For entity linking over the questions, we retrieve the top 20 entities based on the match between a bigram based TFIDF vector of the question with a similar vector derived from the surface form of the entity (same as the title of the Wiki article). We found that the gold entities that need to be retrieved are within 2 hops of the entities linked in this manner for 87% of the dev examples.

Unlike the MetaQA and WikiData datasets, however, for HotpotQA we do not know the number of hops required for each question in advance. Instead, we run DrKIT for 2 hops for each question, and then take a weighted average of the distribution over entities after each hop $Z^* = \pi_0 Z_0 + \pi_1 Z_1 + \pi_2 Z_2$. Z_0 consists of the entities linked to the question itself, rescored based on an encoding of the question, since in some cases one or both the entities to be retrieved are in this set.[10] Z_1 and Z_2 are given by Eq. 8.5. The mixing weights π_i are the softmax outputs of a classifier on top of another encoding of the question, learnt *end-to-end* on the retrieval task. This process can be viewed as soft mixing of different templates ranging from 0 to 2 hops for answering a question, similar to NQL [36].

Results. We compare our retrieval results to those presented in Das et al. [47] in Table 8.4 (Left). We measure the accuracy @k retrievals, which is the fraction of questions for which *both* the required passages are in the top k retrieved ones. We see an improvement in accuracy across the board, with much higher gains @2 and @5. The main baseline is the entity-centric IR approach which runs a BERT-based re-ranker on 200 pairs of passages for each question. Importantly, DrKIT also improves by over 10x in terms of queries per second during inference. Note that the inference time is measured using a batch size of 1 for both models for fair comparison. DrKIT can be easily run with batch sizes up to 40, but the entity centric IR baseline cannot due to the large number of runs of BERT for each query. When comparing different datasets for pretraining the index, there is not much difference between using the WikiData KB, or the

[9] `https://hotpotqa.github.io/wiki-readme.html`

[10] For example, for the question "How are elephants connected to Gajabrishta?", one of the passages to be retrieved is "Gajabrishta" itself.

System	Runtime		Answer		Sup Fact		Joint	
	#Bert	s/Q	EM	F1	EM	F1	EM	F1
Baseline [246]	–	–	25.23	34.40	5.07	40.69	2.63	17.85
Golden Ret [159]	–	1.4†	37.92	48.58	30.69	64.24	18.04	39.13
Semantic Ret [150]	50*	40.0‡	45.32	57.34	38.67	70.83	25.14	47.60
HGN [69]	50*	40.0‡	56.71	69.16	**49.97**	76.39	**35.63**	59.86
Rec Ret [3]	500*	133.2†	**60.04**	**72.96**	49.08	**76.41**	35.35	**61.18**
DrKIT + BERT	**1.2**$^{\diamond}$	**1.3**	42.13	51.72	37.05	59.84	24.69	42.88

Table 8.5: Official leaderboard evaluation on the test set of HotpotQA. #Bert refers to the number of calls to BERT [51] in the model. s/Q denotes seconds per query (using batch size 1) for inference on a single 16-core CPU. Answer, Sup Fact and Joint are the official evaluation metrics for HotpotQA. *: This is the minimum number of BERT calls based on model and hyperparameter descriptions in the respective papers. †: Computed using code released by authors, using a batch size of 1. ‡: Estimated based on the number of BERT calls, using 0.8s as the time for one call (without considering overhead due to other computation in the model). $^{\diamond}$: One call to a 5-layer Transformer, and one call to BERT.

HotpotQA questions. The latter has a better accuracy @2, but overall the best performance is when using a combination of both.

In Table 8.4 (right), we check the performance of the baseline reading comprehension model from Yang et al. [246], when given the passages retrieved by DrKIT. While there is a significant improvement over the baseline which uses a TFIDF based retrieval, we see only a small improvement over the passages retrieved by the entity-centric IR baseline, despite the significantly improved accuracy @10 of DrKIT. Among the 33% questions where the top 10 passages do not contain both the correct passages, for around 20% the passage containing the answer is also missing. We conjecture this percentage is lower for the entity-centric IR baseline, and the downstream model is able to answer some of these questions without the other supporting passage.

8.3.4 HotpotQA: End-to-End Answer Extraction

In this section, we build an end-to-end pipeline for answering HotpotQA questions. We use DrKIT to identify the top 5 passages which are likely to contain the answer to a question. We

then train a separate model to extract the answer from a concatenation of these passages. This model is a standard BERT-based architecture used for SQuAD (see Devlin et al. [51] for details), with a few modifications. First, to handle boolean questions, we train a 3-way classifier on top of the [CLS] representation from BERT to decide whether the question has a "span", "yes" or "no" answer, respectively. During inference, if this classifier has the highest probability on "span" we extract a start and end position similar to Devlin et al. [51], else we directly answer as "yes" or "no".

Second, to handle supporting fact prediction, we prepend each sentence in the concatenated passages passed to BERT with a special symbol [unused0], and train a binary classifier on top of the representation of each of these symbols output from BERT. The binary classifier is trained to predict 1 for sentences which are supporting facts and 0 for sentences which are not. During inference, we take all sentences for which the output probability of this classifier is greater than 0.5 as supporting facts.

The training loss is an average of the loss for the 3-way classifier ($\mathcal{L}_{cls}$), the sum of the losses for the supporting fact classifiers ($\mathcal{L}_{sp}$), and the losses for the start and end positions of span answers ($\mathcal{L}_{st}$, $\mathcal{L}_{en}$):

$$\mathcal{L} = (\mathcal{L}_{cls} + \mathcal{L}_{sp} + \mathcal{L}_{st} + \mathcal{L}_{en})/4 \tag{8.9}$$

We train the system on 5 passages per question provided in the distractor setting of HotpotQA—2 gold ones and 3 negatives from a TFIDF retriever. We keep the gold passages at the beginning for 60% of the examples, and randomly shuffle all passages for the rest, since during inference the correct passages are likely to be retrieved at the top by DrKIT. Other hyperparameters include batch size 32, learning rate 5×10^{-5}, number of training epochs 5, and a maximum combined passage length 512.

Table 8.5 shows the performance of this system on the HotpotQA test set, compared with other recently published models.[11] In terms of accuracy, DrKIT+BERT reaches a modest score of 42.88 joint F1, but is considerably faster (up to 100x) than the models which outperform it.

8.4 Discussion

Neural Query Language (NQL) [36] defines differentiable templates for multi-step access to a symbolic KB, in which relations between entities are *explicitly* enumerated. The motivation behind this work was to extend those ideas to a setting where relations are defined *implicitly*

[11] As of February 23, 2020: https://hotpotqa.github.io/.

in text. We contrast this with the extensive line of work on Knowledge Graph embeddings [20, 50, 241], which also uses neural representations, but derives them from a limited set of relations in a symbolic KB. Such embeddings often allow generalization to unseen facts using relation patterns, but text corpora are more complete in the information they contain.

Talmor and Berant [200] also examined answering compositional questions by treating a text corpus (in their case the entire web) as a KB. However their approach consists of parsing the query into a computation tree, and running a black-box QA model on its leaves separately, which *cannot* be trained end-to-end. This is analogous to semantic parsing, but the logical form derived from the question is executed using an open-domain QA model, rather than a query language. Our approach lies somewhere in between—the sparse entity set and relation vectors we derive from the question effectively form a semantic parse, and the textual follow operation constitutes a simple query language. However, this semantic parse is latent and we show that it can be learned efficiently even against a large knowledge source. An interesting direction of future work would to be integrate this with tasks other than QA, which also need to reason over background knowledge, e.g. language modeling and dialogue.

There is also a question remaining as to whether this approach can work at a much larger web-scale. The largest index we construct consisted of all abstracts on Wikipedia ($\sim$ 34GB). Even extending this to full Wikipedia articles would result in a 5x increase, and extending to the entire web may be infeasible outside industrial settings. Hence, one direction for future work would be to explore methods for compressing the virtual KB—either by reduce the embedding size of the representations, or by removing redundancy, e.g. by clustering the mentions.

Traditional KBs are useful beyond just following paths of relations, and can be used for numerical operations like finding the maximum or minimum value of a property, or counting the number of entities which satisfy a property. Hence, another direction of future work would be to build such operations over the virtual KB.

Chapter 9

Conclusions

In this thesis we studied methods for reading and reasoning over textual knowledge, focusing on factual information, and primarily on Wikipedia, but also on news articles and biomedical abstracts. We developed neural network models for representing text passages as a collection of dense feature vectors which can be used to classify answers to natural language questions. By leveraging external entity linking tools we also incorporated symbolic knowledge graphs into the process for answering the questions. Finally, we proposed a novel representation for textual knowledge in the form of a virtual KB, which offers some of the basic functionality of traditional KBs, such as slot-filling and multi-hop reasoning, but also generalizes to unseen entities and unseen relations. The models presented in this thesis advanced the state-of-the-art for several benchmarks for reading comprehension and open-domain question answering at the time they were introduced.

In this chapter, we start by summarizing the key contributions of this thesis in § 9.1, followed by a discussion of the key ideas in § 9.2. Then we conclude by outlining some directions for future work in § 9.3.

9.1 Summary of Contributions

Reading Comprehension. In Chapter 3 we introduced a novel attention mechanism, called *Gated-Attention*, and incorporated it into a standard neural network architecture based on RNNs. We showed that introducing this attention mechanism leads to a substantial improvement in performance on 5 benchmarks for answering questions against a text passage. We also introduced *Coref-RNNs*, an extension to standard RNNs, which model recurrence both along the input sequence as well as coreference chains detected by an external system. Replacing

standard RNNs with Coref-RNNs in the reading model leads to further improvements on the WikiHop [223], bAbi [227] and LAMBADA [152] datasets, which require aggregating information from multiple sentences in the passage linked together by coreferent mentions. In Chapter 4, we discussed methods for transfer learning from unlabeled text to the reading models. We analyzed several design choices when pretraining word embeddings, and proposed a novel pretraining objective, based on structure of the unlabeled documents, for the other components of the model. We showed that the latter leads to a huge improvement on Squad [166], TriviaQA [104] and BioASQ when the supervised data is limited.

QA over KBs and Text. In Chapter 5 we studied the practical setting of answering questions against an large text corpus and an incomplete KB. We introduced a two-step model, called *Graft-Net*, which consists of a retrieval step followed by a graph-comprehension step. In particular, for the latter step, we adapted a commonly used neural architecture, graph convolution networks, to the open-domain QA setting by: (i) introducing heterogeneous update rules for handling text sentences and KB facts; and (ii) introducing an inductive bias in the form of directed propagation for restricting the model to only encode paths relevant to the question. We showed the effectiveness of the overall pipeline on a several KB settings—ranging from having no KB, to an incomplete KB, to a fully complete one—by comparing to several baselines from prior work on the WikiMovies [135] and WebQuestionsSP [14] benchmarks. We showed that representing the text and KB jointly, which we call *early fusion*, leads to substantial improvements over models which represent them separately.

Multi-turn QA using Reinforcement Learning. In Chapter 6 we developed a model for answering user queries in a multi-turn interaction. We discussed a modular neural network, and proposed to use reinforcement learning for training it from user feedback. We also identified an important limitation—prior approaches for accessing the underlying KB rely on constructing a symbolic query which is not differentiable. We addressed this limitation by developing a soft method for computing a posterior distribution over the KB contents, and used statistics computed from the distribution to decide the next agent action. We also discussed a two-stage training strategy for the agent, consisting of an imitation learning step which emulates a rule-based agent, followed by reinforcement learning based on feedback from a user simulator. We showed gains in succesfully answering queries in fewer turns by using the soft posterior over the KB over the symbolic query approach.

Virtual Knowledge Base. In Chapters 7 we introduced virtual KBs, which store pretrained contextual representations of spans in a corpus. In Chapter 8 we further extended these by linking together the mention spans using cooccurrence and coreference relations in a graph represented as a sparse matrix. We show that virtual KBs can emulate important features of a traditional KB, such as slot-filling and multi-hop reasoning, while maintaining a fast speed of inference. We also showed that virtual KBs can generalize to queries about unseen entities and relations during training, and also allow training a soft semantic parser from denotations. We establish a new state-of-the-art on the MetaQA [260] dataset, and show competitive performance on HotpotQA [246], while being an order of magnitude faster than prior approaches.

9.2 Key Ideas

Architectural Inductive Biases. A recurring theme in the methods presented in this thesis is that we can build inductive biases into deep learning models to better adapt them to certain tasks. An inductive bias is an assumption in the learning algorithm which prioritizes one set of solutions over another [10]. In machine learning, inductive biases can take many forms, such as the prior distribution in bayesian learning, or regularization terms in the loss function. In deep learning, an important type of inductive bias is encoded in the architecture of the neural network itself. The classic example of an architectural inductive bias is the use of convolutions for representing images in CNNs, which lends invariance towards the translation of objects within the image.

Throughout this thesis we have introduced many inductive biases for representing text, knowledge graphs, and answering questions. The Gated-Attention mechanism and Coref-RNNs (Chapter 3) introduce biases for learning query-dependent representations which also encode coreferent-recency. Similarly, the directed propagation method (Chapter 5) biases graph networks for encoding only paths which start at an entity mentioned in the question. Entropy statistics computed over the belief states of KB-InfoBot (Chapter 6) provide another bias for the dialogue policy network to not overfit on specific values of the distributions.

Graph Structures for Reasoning. Reasoning can be loosely defined as the ability to derive new information from what is known. Reasoning over text often involves aggregating information from multiple passages. To enable this aggregation, at various points in this thesis we relied on connecting text spans across passages in a graph structure using entity linking and coreference relations. Depending on the setting, we introduced several methods for representing the

resulting graph structure.

In Chapter 3, after augmenting a text sequence with coreference relations we end up with two Directed-Acyclic Graphs (DAGs), one from left to right and another from right to left in the sequence. Coref-RNNs generalize the idea of recurrence to DAGs, while being more efficient than general graph networks. In Chapter 5, after linking text passages to a KB, we end up with a general graph with cycles, for which we used the more general graph network approach. The virtual KB augmented with entity structure in Chapter 8 can be viewed as a bi-partite graph, with text passages on one side linked to entities on the other. The textual-follow operation introduced there essentially hops from entities to passages with which they are connected, and then back to co-occurring entities, with an inner-product based scoring step in between. In each of these cases, the graph structure provides an inductive bias for learning representations suited for reasoning across text passages.

Differentiable Computation. Deep learning relies on stochastic gradient descent for optimizing the parameters of neural networks which generate representations of data. However, operations for manipulating structured data, e.g. running SQL queries, are not differentiable, and hence training networks which learn to execute such operations is challenging. Hence, in this thesis we develop differentiable counterparts to a few such operations. The basic idea behind these counterparts is to replace the discrete results of the operation by a soft probability distribution over all possible results, such that the weights of the distribution are computed in a differentiable manner from the inputs of the operation.

One such counterpart was introduced in Chapter 6, where we replaced select SQL statements against a table with the soft-KB lookup. Another counterpart was introduced in Chapter 8, where we replaced the discrete relation-following operation with a softer version in DrKIT. These ideas contribute to the broader theme of *neural-symbolic* learning, an ongoing research effort towards combining symbolic logic and deep learning [17].

9.3 Future Work

The thesis of this research has been that, through the use of machine learning, text corpora can replace knowledge bases for serving information needs. While we have made much progress, realizing this goal requires extending existing methods and addressing several limitations, which we highlight here.

Beyond Question Answering. Throughout, our focus has been on various forms of QA. However, traditional KBs are used for all sorts of automated decision making tasks. In particular, it will be interesting to see whether the virtual KB framework can be integrated with end-to-end learning for tasks which implicitly depend on real-world knowledge, e.g. detecting fake news, intelligent tutoring, and automatic customer support. As a concrete next step, we can extend state-of-the-art language models [51, 86] with an operation which can construct queries to the virtual KB, and use the retrieved facts to predict the next token.

Other types of Reasoning. The DrKIT model introduced in Chapter 8 answers natural language questions by parsing them to soft logical forms and then executing the logical forms against the virtual KB. However, the logical forms were restricted to relatively simple templates, e.g. in the 2-hop case $\lambda x.\exists y R_1(e, y)R_2(y, x)$. This limits the class of questions our system can handle, and more complex information needs require executing richer logical forms. In particular, an important direction for future work is to develop differentiable implementations of reasoning operations beyond relation following. We outline some ideas below:

1. **Set intersection & union:** Many questions involve taking an intersection or union of sets of intermediate answers, e.g. "Which movies have starred both Tom Hanks and Julia Roberts?", or "Which Pittsburgh restaurants serve either Chinese or Indian food?". These operations can be easily implemented using the sparse vector representation for weighted sets of entities introduced in Chapter 8. Suppose v_{X_1} and v_{X_2} are sparse vectors for sets X_1 and X_2. Then we can model their intersection as $v_{X_1 \cap X_2} = v_{X_1} \odot v_{X_2}$ ($\odot$ is an elementwise product), and union as $v_{X_1 \cup X_2} = \max(v_{X_1}, v_{X_2})$ or $v_{X_1 \cup X_2} = v_{X_1} + v_{X_2}$ [36].
2. **N-ary relation following:** Real-world KBs encode rich N-ary relations with qualifiers, such as start and end times, over predicates. This is useful for answering questions like "Which club did Cristiano Ronaldo play for *in 2007*?". One way to answer such queries using text is to redefine the relation to a more fine-grained version (e.g. *member-of-club-in-2007* as opposed to *member-of-club*), and, in fact, this may be learned end-to-end by the relation-following operation. But a more principled approach would allow following a *specified* slot starting from an arbitrary number of entities which fill other slots in an N-ary relation.
3. **Mathematical reasoning:** In this thesis we ignored an important class of numerical operations which are useful for answering questions such as "Which was the *last* album released by Metallica?". One approach involves augmenting a neural model with a set of

discrete operations, and searching among them for a sequence which produces the correct answer [2, 61]. Differentiable implementations of these operations would allow searching in an end-to-end manner without relying on heuristics which are commonly employed to prune the search space. While recent work [84] has explored such implementations, these are currently limited to at most a few text passages.

4. **Negation:** Certain questions require identifying sets of entities which *do not* satisfy a relation, e.g. "Which platinum albums have not won a Grammy?". If every negated instance was explicitly written in the text, we may have answered this by conceiving of a negated relation type (e.g. *not-awarded-prize*), but this will rarely be the case. When dealing with small sets of entities, we may be able to invert the confidence scores of the set of entities which do satisfy the relation. But recall that, in DrKIT, we only retrieved the top K entities which satisfied a relation. Hence, this naïve approach will treat all other entities outside the top K as equally likely to not satisfy the relation. It is not clear at this point how these difficulties may be addressed.

Other types of Knowledge. In this thesis we have largely modeled explicit factual knowledge, the kind which appears in encyclopedias or news articles, and which can typically be expressed as relations and attributes of entities. Other forms of knowledge are more *implicit*, such as commonsense or social and cultural norms, and this is usually harder to express in terms of entities or relations. This kind of knowledge is best expressed in natural language statements, but is rarely found in encyclopedias. Instead, an intriguing line of work has recently discussed using language models, trained on web-scale corpora, as repositories of commonsense knowledge [165, 206].

Language models have also been touted as KBs themselves, since they exhibit some capacity to encode common facts [158]. But they also have limitations – (i) different queries which are semantically the same can lead to different answers from LMs, demonstrating their brittleness;[1] (ii) they suffer with queries which involve reasoning [201]; and (iii) it is not clear how to add or remove facts from LMs, since it is not clear where they derive their knowledge from in the first place. The explicit methods for representing knowledge discussed in this thesis do not suffer from these limitations, and hence combining the two approaches has potential.

[1]This can be easily verified by playing around with one: `https://transformer.huggingface.co/doc/gpt2-large`.

Uncertain & Dynamic Knowledge. The methods discussed here all assume an unchanging corpus from when training the model to when it is tested. We also assumed that the queries of interest have a fixed unambiguous answer given the knowledge source. Practical situations in which users attempt to access knowledge, however, are much more dynamic and uncertain. For example, noteworthy news events, such as Donald Trump's impeachment trial, or the COVID-19 pandemic, are recorded in corpora where information changes from day-to-day. Building a question answering system over such corpora involves unique challenges, such as aggregating contradictory information, modeling trustworthiness of sources, and distribution shift from train to test, to name a few. Tackling these issues requires new models, but also new benchmarks which go beyond the static setting of existing datasets.

Of particular importance is the ability to model the *confidence* of an extracted answer, a relatively understudied aspect of text-based QA. Typically in machine learning, confidence is estimated as the posterior probability of a prediction given the input, and in supervised learning this is modeled directly. Clark and Gardner [31] discuss several strategies for calibrating the confidence scores of answers extracted from multiple documents. The underlying documents themselves are assumed to be completely correct, and the only source of error is the model itself. In a practical setting, however, there may be an error associated with the documents as well (e.g. incorrect edits on Wikipedia, misinformation in news). One way to account for this error is to associate a prior probability to each document in the corpus, which roughly captures the truthfulness of its information. This is similar in spirit to *probabilistic databases* [193], which associate to each fact in the KB a probability whether it is correct or not, and during inference propagate it to the confidence of the final answer. Developing a similar framework when using text as a knowledge source is an important direction for future research.

Bibliography

[1] Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. Practical and optimal lsh for angular distance. In *Advances in Neural Information Processing Systems*, pages 1225–1233, 2015. 1.1, 2.2.3, 8.2.2

[2] Daniel Andor, Luheng He, Kenton Lee, and Emily Pitler. Giving bert a calculator: Finding operations and arguments with reading comprehension. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5949–5954, 2019. 3

[3] Akari Asai, Kazuma Hashimoto, Hannaneh Hajishirzi, Richard Socher, and Caiming Xiong. Learning to retrieve reasoning paths over wikipedia graph for question answering. In *International Conference on Learning Representations*, 2020. 8.3.4

[4] Layla El Asri, Jing He, and Kaheer Suleman. A sequence-to-sequence model for user simulation in spoken dialogue systems. *arXiv preprint arXiv:1607.00070*, 2016. 6.5.1

[5] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In Karl Aberer, Key-Sun Choi, Natasha Noy, Dean Allemang, Kyung-Il Lee, Lyndon Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Riichiro Mizoguchi, Guus Schreiber, and Philippe Cudré-Mauroux, editors, *The Semantic Web*, pages 722–735, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. ISBN 978-3-540-76298-0. 5.1

[6] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 2.2.1

[7] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014. 1.1, 3.1

[8] Ondrej Bajgar, Rudolf Kadlec, and Jan Kleindienst. Embracing data abundance: Booktest dataset for reading comprehension. *arXiv preprint arXiv:1610.00956*, 2016. (document),

3.4.1, 3.4.1, 3.5.1, 4.1

[9] Collin F Baker, Charles J Fillmore, and John B Lowe. The berkeley framenet project. In *Proceedings of the 17th international conference on Computational linguistics-Volume 1*, pages 86–90. Association for Computational Linguistics, 1998. 2.1.1

[10] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018. 9.2

[11] Petr Baudiš. Yodaqa: a modular question answering system pipeline. In *POSTER 2015-19th International Student Conference on Electrical Engineering*, 2015. 5.1

[12] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994. 1.1, 2.2.1

[13] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003. 1

[14] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on Freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544, Seattle, Washington, USA, October 2013. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/D13-1160`. 2.1.3, 9.1

[15] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544, 2013. 1

[16] Tim Berners-Lee, James Hendler, Ora Lassila, et al. The semantic web. *Scientific american*, 284(5):28–37, 2001. 1

[17] Tarek R Besold, Artur d'Avila Garcez, Sebastian Bader, Howard Bowman, Pedro Domingos, Pascal Hitzler, Kai-Uwe Kühnberger, Luis C Lamb, Daniel Lowd, Priscila Machado Vieira Lima, et al. Neural-symbolic learning and reasoning: A survey and interpretation. *arXiv preprint arXiv:1711.03902*, 2017. 9.2

[18] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a

collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250. AcM, 2008. 1, 2.1.2, 2, 5.1, 5.4.1

[19] Antoine Bordes and Jason Weston. Learning end-to-end goal-oriented dialog. *arXiv preprint arXiv:1605.07683*, 2016. 6.6

[20] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Advances in neural information processing systems*, pages 2787–2795, 2013. 8.4

[21] Antoine Bordes, Nicolas Usunier, Sumit Chopra, and Jason Weston. Large-scale simple question answering with memory networks. *arXiv preprint arXiv:1506.02075*, 2015. 2.1.3

[22] Ronald Brachman and Hector Levesque. Knowledge representation and reasoning. 2004. 2.1.1

[23] Devendra Singh Chaplot, Kanthashree Mysore Sathyendra, Rama Kumar Pasumarthi, Dheeraj Rajagopal, and Ruslan Salakhutdinov. Gated-attention architectures for task-oriented language grounding. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. 3.6

[24] Eugene Charniak. *Toward a model of children's story comprehension.* PhD thesis, Massachusetts Institute of Technology, 1972. 1.1

[25] Danqi Chen, Jason Bolton, and Christopher D Manning. A thorough examination of the cnn/daily mail reading comprehension task. In *ACL*, 2016. 3.1, 3.4.1, 3.5.1, 4.1, 4.2.1, 4.2.4

[26] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading wikipedia to answer open-domain questions. In *ACL*, 2017. (document), 1.1, 2.1.3, 2.2.3, 3.6, 5.3, 5.4.2, 8.1, 8.1, 8.3.2

[27] Huadong Chen, Shujian Huang, David Chiang, and Jiajun Chen. Improved neural machine translation with a syntax-aware encoder and decoder. *ACL*, 2017. 3.5.2

[28] Yun-Nung Chen, Dilek Hakkani-Tür, Gokhan Tur, Jianfeng Gao, and Li Deng. End-to-end memory networks with knowledge carryover for multi-turn spoken language understanding. In *Proceedings of The 17th Annual Meeting of the International Speech Communication Association*, 2016. 6.3

[29] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using

rnn encoder-decoder for statistical machine translation. *ACL*, 2015. 2.2.1

[30] Zewei Chu, Hai Wang, Kevin Gimpel, and David McAllester. Broad context language modeling as reading comprehension. *EACL*, 2017. 3.4.2, 3.4.2, 3.4.2

[31] Christopher Clark and Matt Gardner. Simple and effective multi-paragraph reading comprehension. In *ACL*, 2018. 3.6, 4.1, 4.3.1, 4.3.2, 6, 9.3

[32] Kevin Clark and Christopher D Manning. Entity-centric coreference resolution with model stacking. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pages 1405–1415, 2015. 3.4.2

[33] Peter Clark and Oren Etzioni. My computer is an honor student—but how intelligent is it? standardized tests as a measure of ai. *AI Magazine*, 37(1):5–12, 2016. 3

[34] William W Cohen. *Graph walks and graphical models*, volume 5. Citeseer, 2010. 2.2.3

[35] William W Cohen. Tensorlog: A differentiable deductive database. *arXiv preprint arXiv:1605.06523*, 2016. 2.1.2

[36] William W Cohen, Matthew Siegler, and Alex Hofer. Neural query language: A knowledge base query language for tensorflow. *arXiv preprint arXiv:1905.06209*, 2019. 2.1.2, 8.1, 8.3.3, 8.4, 1

[37] National Research Council. *Defining a Decade: Envisioning CSTB039;s Second 10 Years*. The National Academies Press, Washington, DC, 1997. ISBN 978-0-309-05933-6. doi: 10.17226/5903. URL `https://www.nap.edu/catalog/5903/defining-a-decade-envisioning-cstbs-second-10-years`. 1

[38] Heriberto Cuayáhuitl, Steve Renals, Oliver Lemon, and Hiroshi Shimodaira. Human-computer dialogue simulation using hidden markov models. In *Automatic Speech Recognition and Understanding, 2005 IEEE Workshop on*, pages 290–295. IEEE, 2005. 6.5.1

[39] Yiming Cui, Zhipeng Chen, Si Wei, Shijin Wang, Ting Liu, and Guoping Hu. Attention-over-attention neural networks for reading comprehension. *ACL*, 2017. 3.4.1, 3.5.1

[40] Michał Daniluk, Tim Rocktäschel, Johannes Welbl, and Sebastian Riedel. Frustratingly short attention spans in neural language modeling. *ICLR*, 2017. 3.1

[41] Rajarshi Das, Arvind Neelakantan, David Belanger, and Andrew McCallum. Chains of reasoning over entities, relations, and text using recurrent neural networks. In *EACL*, 2016. 2.1.2

[42] Rajarshi Das, Shehzaad Dhuliawala, Manzil Zaheer, Luke Vilnis, Ishan Durugkar, Akshay Krishnamurthy, Alex Smola, and Andrew McCallum. Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning. *arXiv preprint arXiv:1711.05851*, 2017. (document), 2.1.3, 5.3

[43] Rajarshi Das, Arvind Neelakantan, David Belanger, and Andrew McCallum. Chains of reasoning over entities, relations, and text using recurrent neural networks. In *EACL*, 2017. 2.1.2

[44] Rajarshi Das, Manzil Zaheer, Siva Reddy, and Andrew McCallum. Question answering on knowledge bases and text using universal schema and memory networks. In *ACL*, 2017. 2.1.3, 5.1, 5.4.2

[45] Rajarshi Das, Shehzaad Dhuliawala, Manzil Zaheer, Luke Vilnis, Ishan Durugkar, Akshay Krishnamurthy, Alex Smola, and Andrew McCallum. Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning. In *ICLR*, 2018. 2.1.2

[46] Rajarshi Das, Shehzaad Dhuliawala, Manzil Zaheer, and Andrew McCallum. Multi-step retriever-reader interaction for scalable open-domain question answering. In *ICLR*, 2019. (document), 7.2.2, 8.3

[47] Rajarshi Das, Ameya Godbole, Dilip Kavarthapu, Zhiyu Gong, Abhishek Singhal, Mo Yu, Xiaoxiao Guo, Tian Gao, Hamed Zamani, Manzil Zaheer, and Andrew McCallum. Multi-step entity-centric information retrieval for multi-hop question answering. In *Proceedings of the 2nd Workshop on Machine Reading for Question Answering*, pages 113–118, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-5816. URL `https://www.aclweb.org/anthology/D19-5816`. (document), 8.4, 8.3.3, 8.3.3

[48] Pradeep Dasigi, Nelson F. Liu, Ana Marasović, Noah A. Smith, and Matt Gardner. Quoref: A reading comprehension dataset with questions requiring coreferential reasoning. In *Proc. of EMNLP-IJCNLP*, 2019. 3.6

[49] Randall Davis, Howard Shrobe, and Peter Szolovits. What is a knowledge representation? *AI magazine*, 14(1):17, 1993. 1

[50] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. 8.4

[51] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, 2018. (document), 1.1, 3.6, 1, 4.3.2, 4.4, 7.1, 7.2.2, 7.2.2, 8.1, 8.2.2, 8.5, 8.3.4, 9.3

[52] Bhuwan Dhingra, Zhong Zhou, Dylan Fitzpatrick, Michael Muehl, and William W. Cohen. Tweet2vec: Character-based distributed representations for social media. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 2: Short Papers*, 2016. URL `http://aclweb.org/anthology/P/P16/P16-2044.pdf`. 3.2

[53] Bhuwan Dhingra, Lihong Li, Xiujun Li, Jianfeng Gao, Yun-Nung Chen, Faisal Ahmed, and Li Deng. Towards end-to-end reinforcement learning of dialogue agents for information access. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 484–495, 2017. 1.1

[54] Bhuwan Dhingra, Hanxiao Liu, Zhilin Yang, William Cohen, and Ruslan Salakhutdinov. Gated-attention readers for text comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1832–1846, 2017. 1.1, 4.1

[55] Bhuwan Dhingra, Kathryn Mazaitis, and William W Cohen. Quasar: Datasets for question answering by search and reading. *arXiv preprint arXiv:1707.03904*, 2017. 1.1, 3.6, 5.2

[56] Bhuwan Dhingra, Danish Danish, and Dheeraj Rajagopal. Simple and effective semi-supervised question answering. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 582–587, 2018. 1.1

[57] Bhuwan Dhingra, Qiao Jin, Zhilin Yang, William Cohen, and Ruslan Salakhutdinov. Neural models for reasoning over multiple mentions using coreference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 42–48, 2018. 1.1

[58] Bhuwan Dhingra, Danish Pruthi, and Dheeraj Rajagopal. Simple and effective semi-supervised question answering. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 2 (Short*

Papers), pages 582–587, 2018. URL https://aclanthology.info/papers/N18-2092/n18-2092. 5.1

[59] Bhuwan Dhingra, Manzil Zaheer, Vidhisha Balachandran, Graham Neubig, Ruslan Salakhutdinov, and William W Cohen. Differentiable reasoning over a virtual knowledge base. In *Proceedings of the International Conference on Learning Representations*, 2020. 1.1

[60] Ming Ding, Chang Zhou, Qibin Chen, Hongxia Yang, and Jie Tang. Cognitive graph for multi-hop reading comprehension at scale. In *ACL*, 2019. 5.5

[61] Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *Proc. of NAACL*, 2019. 3.6, 3

[62] Greg Durrett and Dan Klein. Easy victories and uphill battles in coreference resolution. In *EMNLP*, pages 1971–1982, 2013. 3.1, 3.4.2

[63] Greg Durrett and Dan Klein. A joint model for entity analysis: Coreference, typing, and linking. *Transactions of the association for computational linguistics*, 2:477–490, 2014. 1.1

[64] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pages 2224–2232, 2015. 2.2.2

[65] Chris Dyer. Should neural network architecture reflect linguistic structure? CoNLL Keynote, 2017. URL http://www.conll.org/keynotes-2017. 3.1

[66] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990. 2.2.1

[67] Mihail Eric, Lakshmi Krishnan, Francois Charette, and Christopher D Manning. Key-value retrieval networks for task-oriented dialogue. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, pages 37–49, 2017. 6.7

[68] Oren Etzioni, Michele Banko, and Michael J Cafarella. Machine reading. In *AAAI*, volume 6, pages 1517–1519, 2006. 3

[69] Yuwei Fang, Siqi Sun, Zhe Gan, Rohit Pillai, Shuohang Wang, and Jingjing Liu. Hierarchical graph network for multi-hop question answering. *arXiv preprint arXiv:1911.03631*, 2019. 8.3.4

[70] Paolo Ferragina and Ugo Scaiella. Fast and accurate annotation of short texts with

wikipedia pages. *IEEE software*, 29(1):70–75, 2012. 1, 2.2.3

[71] David Ferrucci, Eric Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya A Kalyanpur, Adam Lally, J William Murdock, Eric Nyberg, John Prager, et al. Building watson: An overview of the deepqa project. *AI magazine*, 2010. 2.1.3, 3, 5.1

[72] Charles J Fillmore. Frame semantics and the nature of language. *Annals of the New York Academy of Sciences*, 280(1):20–32, 1976. 2.1.1

[73] Matt Gardner and Jayant Krishnamurthy. Open-vocabulary semantic parsing with both distributional statistics and formal knowledge. In *AAAI*, pages 3195–3201, 2017. 2.1.3

[74] M Gašić, Catherine Breslin, Matthew Henderson, Dongho Kim, Martin Szummer, Blaise Thomson, Pirros Tsiakoulis, and Steve Young. On-line policy optimisation of bayesian spoken dialogue systems via human interaction. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8367–8371. IEEE, 2013. 6.1

[75] Lise Getoor and Ben Taskar. *Introduction to statistical relational learning*. MIT press, 2007. 2.1.2

[76] Zoubin Ghahramani and Katherine A Heller. Bayesian sets. In *Advances in neural information processing systems*, pages 435–442, 2006. 7.2.4

[77] Daniel Gillick, Alessandro Presta, and Gaurav Singh Tomar. End-to-end retrieval in continuous space. *arXiv preprint arXiv:1811.08008*, 2018. 7.1

[78] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. *ICML*, 2017. 2.2.2

[79] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. Similarity search in high dimensions via hashing. In *Vldb*, volume 99, pages 518–529, 1999. 2.2.3

[80] Peter W Glynn. Likelihood ratio gradient estimation for stochastic systems. *Communications of the ACM*, 33(10):75–84, 1990. 6.4

[81] Yichen Gong and Samuel R Bowman. Ruminating reader: Reasoning with gated multi-hop attention. *arXiv preprint arXiv:1704.07415*, 2017. 2.1.3

[82] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014. 2.1.1

[83] Evan Greensmith, Peter L Bartlett, and Jonathan Baxter. Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, 5(Nov):1471–1530, 2004. 6.4

[84] Nitish Gupta, Kevin Lin, Dan Roth, Sameer Singh, and Matt Gardner. Neural module networks for reasoning over text. *arXiv preprint arXiv:1912.04971*, 2019. 3

[85] Kelvin Guu, John Miller, and Percy Liang. Traversing knowledge graphs in vector space. In *EMNLP*, 2015. 2.1.2

[86] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. Realm: Retrieval-augmented language model pre-training. *arXiv preprint arXiv:2002.08909*, 2020. 4.4, 5.5, 9.3

[87] Dilek Hakkani-Tür, Gokhan Tur, Asli Celikyilmaz, Yun-Nung Chen, Jianfeng Gao, Li Deng, and Ye-Yi Wang. Multi-domain joint semantic frame parsing using bi-directional RNN-LSTM. In *Proceedings of The 17th Annual Meeting of the International Speech Communication Association*, 2016. 6.3

[88] Xu Han, Zhiyuan Liu, and Maosong Sun. Joint representation learning of text and knowledge for knowledge graph completion. *arXiv preprint arXiv:1611.04125*, 2016. 2.1.2

[89] Taher H Haveliwala. Topic-sensitive pagerank. In *Proceedings of the 11th international conference on World Wide Web*, pages 517–526. ACM, 2002. 2.2.3, 5.1

[90] Patrick J Hayes. The logic of frames. In *Readings in artificial intelligence*, pages 451–458. Elsevier, 1981. 2.1.1

[91] Mikael Henaff, Jason Weston, Arthur Szlam, Antoine Bordes, and Yann LeCun. Tracking the world state with recurrent entity networks. *arXiv preprint arXiv:1612.03969*, 2016. 3.3, 3.4.2, 3.4.2, 3.5.2

[92] Matthew Henderson. Machine learning for dialog state tracking: A review. *Machine Learning in Spoken Language Processing Workshop*, 2015. 6.3

[93] Matthew Henderson, Blaise Thomson, and Steve Young. Word-based dialog state tracking with recurrent neural networks. In *Proceedings of the 15th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*, pages 292–299, 2014. 6.3, 6.3

[94] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *Advances in neural information processing systems*, pages 1693–1701, 2015. 3.1, 3.4.1, 3.4.1, 3.5.1

[95] Felix Hill, Antoine Bordes, Sumit Chopra, and Jason Weston. The goldilocks principle: Reading children's books with explicit memory representations. *ICLR*, 2016. 3.1, 3.4.1,

3.4.1, 3.5.1, 4.2.1

[96] Geoffrey Hinton, N Srivastava, and Kevin Swersky. Lecture 6a overview of mini–batch gradient descent. *Coursera Lecture slides https://class. coursera. org/neuralnets-2012-001/lecture,[Online*, 2012. 6.4

[97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 2.2.1, 3.2

[98] Minghao Hu, Yuxing Peng, and Xipeng Qiu. Mnemonic reader for machine comprehension. *arXiv preprint arXiv:1705.02798*, 2017. 2.1.3

[99] Peter Jackson. *Introduction to expert systems*. Addison-Wesley Longman Publishing Co., Inc., 1998. 2.1.1

[100] Sarthak Jain. Question answering over knowledge base using factual memory networks. In *Proceedings of the NAACL Student Research Workshop*, pages 109–115, 2016. 2.1.3

[101] Yangfeng Ji, Chenhao Tan, Sebastian Martschat, Yejin Choi, and Noah A Smith. Dynamic entity representations in neural language models. *EMNLP*, 2017. 3.5.2

[102] Robin Jia and Percy Liang. Adversarial examples for evaluating reading comprehension systems. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2017. 1.1, 3.6, 5.1

[103] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *arXiv preprint arXiv:1702.08734*, 2017. 1.1, 7.2.3, 8.2, 8.2.2

[104] Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, Vancouver, Canada, July 2017. Association for Computational Linguistics. 2.1.3, 3.1, 3.6, 4.1, 4.3.2, 9.1

[105] Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S Weld, Luke Zettlemoyer, and Omer Levy. Spanbert: Improving pre-training by representing and predicting spans. *Transactions of the Association for Computational Linguistics*, 8:64–77, 2020. 4.4

[106] Rudolf Kadlec, Martin Schmid, Ondrej Bajgar, and Jan Kleindienst. Text understanding with the attention sum reader network. *ACL*, 2016. 3.1, 3.2, 3.2, 3.4.1, 3.5.1

[107] Divyansh Kaushik and Zachary C Lipton. How much reading does reading comprehension require? a critical investigation of popular benchmarks. In *EMNLP*, 2018. 9

[108] Michael Kifer, Georg Lausen, and James Wu. Logical foundations of object-oriented and

frame-based languages. *Journal of the ACM (JACM)*, 42(4):741–843, 1995. 2.1.1

[109] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 3.4.1, 4.2.1

[110] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. 2017. 2.1.3, 2.2.2, 2.2.2, 5.1

[111] Ryan Kiros, Richard Zemel, and Ruslan R Salakhutdinov. A multiplicative model for learning distributed text-based attribute representations. In *Advances in Neural Information Processing Systems*, pages 2348–2356, 2014. 3.2

[112] Sosuke Kobayashi, Ran Tian, Naoaki Okazaki, and Kentaro Inui. Dynamic entity representations with max-pooling improves machine reading. In *NAACL-HLT*, 2016. 3.4.1, 3.5.1

[113] Stanley Kok and Pedro Domingos. Statistical predicate invention. In *ICML*, 2007. 2.1.2

[114] Tom Kwiatkowski, Eunsol Choi, Yoav Artzi, and Luke Zettlemoyer. Scaling semantic parsers with on-the-fly ontology matching. In *EMNLP*, 2013. 2.1.3

[115] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466, 2019. 2.1.3, 3.6

[116] Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. Race: Large-scale reading comprehension dataset from examinations. *arXiv preprint arXiv:1704.04683*, 2017. 3.6

[117] Ni Lao, Tom Mitchell, and William Cohen. Random walk inference and learning in a large scale knowledge base. In *EMNLP*, 2011. 2.1.2

[118] Ni Lao, Amarnag Subramanya, Fernando Pereira, and William W Cohen. Reading the web with learned syntactic-semantic inference rules. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1017–1026. Association for Computational Linguistics, 2012. 2.1.2

[119] Kenton Lee, Luheng He, Mike Lewis, and Luke Zettlemoyer. End-to-end neural coreference resolution. *EMNLP*, 2017. 3.1

[120] Hector J Levesque and Ronald J Brachman. Expressiveness and tractability in knowledge

representation and reasoning 1. *Computational intelligence*, 3(1):78–93, 1987. 2.1.1

[121] Omer Levy, Yoav Goldberg, and Ido Dagan. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225, 2015. 4.2.2, 4.2.3

[122] Omer Levy, Minjoon Seo, Eunsol Choi, and Luke Zettlemoyer. Zero-shot relation extraction via reading comprehension. In *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017)*, pages 333–342, 2017. 7.1, 7.2.3, 7.2.4, 7.3.1, 7.3.1, 7.3.1, 7.3.2, 7.3.2, 7.3.3, 7.3.4, 7.4, 8.3.2

[123] Jiwei Li, Will Monroe, Alan Ritter, Michel Galley, Jianfeng Gao, and Dan Jurafsky. Deep reinforcement learning for dialogue generation. *EMNLP*, 2016. 6.1

[124] Peng Li, Wei Li, Zhengyan He, Xuguang Wang, Ying Cao, Jie Zhou, and Wei Xu. Dataset and neural recurrent sequence labeling model for open-domain factoid question answering. *arXiv preprint arXiv:1607.06275*, 2016. 3.2

[125] Xin Li and Dan Roth. Learning question classifiers. In *Proceedings of the 19th international conference on Computational linguistics-Volume 1*, pages 1–7. Association for Computational Linguistics, 2002. 4.3.3

[126] Xiujun Li, Zachary C Lipton, Bhuwan Dhingra, Lihong Li, Jianfeng Gao, and Yun-Nung Chen. A user simulator for task-completion dialogues. *arXiv preprint arXiv:1612.05688*, 2016. 6.1, 6.5.1

[127] Xuijun Li, Yun-Nung Chen, Lihong Li, and Jianfeng Gao. End-to-end task-completion neural dialogue systems. *arXiv preprint arXiv:1703.01008*, 2017. 6.6

[128] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *ICLR*, 2016. 2.2.2

[129] Chen Liang, Jonathan Berant, Quoc Le, Kenneth D Forbus, and Ni Lao. Neural symbolic machines: Learning semantic parsers on freebase with weak supervision. *ACL*, 2017. (document), 1, 2.1.3, 5.4.1, 5.3, 5.4.2

[130] Percy Liang and Christopher Potts. Bringing machine learning and compositional semantics together. *Annu. Rev. Linguist.*, 1(1):355–376, 2015. 2.1.3

[131] Xi Victoria Lin, Richard Socher, and Caiming Xiong. Multi-hop knowledge graph reasoning with reward shaping. In *EMNLP*, 2018. 2.1.2

[132] Robert Logan, Nelson F Liu, Matthew E Peters, Matt Gardner, and Sameer Singh. Barack's

wife hillary: Using knowledge graphs for fact-aware language modeling. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5962–5971, 2019. 5.5

[133] Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. Learned in translation: Contextualized word vectors. *arXiv preprint arXiv:1708.00107*, 2017. 4.4

[134] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013. 4.1, 4.2.2

[135] Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. Key-value memory networks for directly reading documents. *arXiv preprint arXiv:1606.03126*, 2016. (document), 2.1.3, 3.3, 5.1, 5.4.1, 5.4.2, 5.3, 8.3.1, 9.1

[136] Bonan Min, Ralph Grishman, Li Wan, Chang Wang, and David Gondek. Distant supervision for relation extraction with an incomplete knowledge base. In *NAACL*, 2013. 1, 2.1.2, 5.1, 7.1

[137] Sewon Min, Danqi Chen, Hannaneh Hajishirzi, and Luke Zettlemoyer. A discrete hard em approach for weakly supervised question answering. In *EMNLP*, 2019. 8.2.1

[138] Sewon Min, Danqi Chen, Luke Zettlemoyer, and Hannaneh Hajishirzi. Knowledge guided text retrieval and reading for open domain question answering. *arXiv preprint arXiv:1911.03868*, 2019. 5.5

[139] Pasquale Minervini, Matko Bošnjak, Tim Rocktäschel, Sebastian Riedel, and Edward Grefenstette. Differentiable reasoning on large knowledge bases and natural language. In *AAAI*, 2020. 2.1.2

[140] Marvin Minsky. A framework for representing knowledge. 1974. 2.1.1

[141] Jeff Mitchell and Mirella Lapata. Vector-based models of semantic composition. In *ACL*, pages 236–244, 2008. 2.1.2, 3.2

[142] Makoto Miwa and Mohit Bansal. End-to-end relation extraction using lstms on sequences and tree structures. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1105–1116, 2016. 2.1.2

[143] Dan Moldovan, Marius Pasca, Sanda Harabagiu, and Mihai Surdeanu. Performance issues and error analysis in an open-domain question answering system. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 33–40,

Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics. doi: 10.3115/1073083.1073091. URL https://www.aclweb.org/anthology/P02-1005. 2.1.3

[144] Diego Moussallem, Mihael Arčan, Axel-Cyrille Ngonga Ngomo, and Paul Buitelaar. Augmenting neural machine translation with knowledge graphs. *arXiv preprint arXiv:1902.08816*, 2019. 5.5

[145] Tsendsuren Munkhdalai and Hong Yu. Neural semantic encoders. *EACL*, 2017. 3.4.1, 3.4.1, 3.5.1

[146] Tsendsuren Munkhdalai and Hong Yu. Reasoning with memory augmented neural networks for language comprehension. *ICLR*, 2017. 3.4.1

[147] Arvind Neelakantan, Benjamin Roth, and Andrew McCallum. Compositional vector space models for knowledge base completion. In *ACL*, 2015. 2.1.2

[148] Allen Newell and Herbert A. Simon. The logic theory machine-a complex information processing system. *IRE Trans. Information Theory*, 2:61–79, 1956. 2.1.1

[149] Allen Newell, John C Shaw, and Herbert A Simon. Report on a general problem solving program. In *IFIP congress*, volume 256, page 64. Pittsburgh, PA, 1959. 2.1.1

[150] Yixin Nie, Songhe Wang, and Mohit Bansal. Revealing the importance of semantic retrieval for machine reading at scale. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2553–2566, 2019. 8.3.4

[151] Takeshi Onishi, Hai Wang, Mohit Bansal, Kevin Gimpel, and David McAllester. Who did what: A large-scale person-centered cloze dataset. *EMNLP*, 2016. 3.1, 3.4.1, 3.4.1, 4.1, 4.2.1

[152] Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The lambada dataset: Word prediction requiring a broad discourse context. *ACL*, 2016. 3.4.2, 9.1

[153] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. *ICML (3)*, 28:1310–1318, 2013. 3.4.1

[154] Nanyun Peng, Hoifung Poon, Chris Quirk, Kristina Toutanova, and Wen-tau Yih. Cross-sentence n-ary relation extraction with graph lstms. *Transactions of the Association for Computational Linguistics*, 5:101–115, 2017. 3.5.2

[155] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL http://www.aclweb.org/anthology/D14-1162. 3.4.1, 3.4.1, 3.4.2, 4.1, 4.2.2

[156] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018. 1.1, 1, 4.4

[157] Matthew E Peters, Mark Neumann, Robert Logan, Roy Schwartz, Vidur Joshi, Sameer Singh, and Noah A Smith. Knowledge enhanced contextual word representations. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 43–54, 2019. 5.5

[158] Fabio Petroni, Tim Rocktäschel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, Alexander H Miller, and Sebastian Riedel. Language models as knowledge bases? 2019. 9.3

[159] Peng Qi, Xiaowen Lin, Leo Mehr, Zijian Wang, and Christopher D. Manning. Answering complex open-domain questions through iterative query generation. In *2019 Conference on Empirical Methods in Natural Language Processing and 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019. URL https://nlp.stanford.edu/pubs/qi2019answering.pdf. (document), 8.4, 8.3.3, 8.3.4

[160] Feng Qian, Lei Sha, Baobao Chang, Lu-chen Liu, and Ming Zhang. Syntax aware lstm model for chinese semantic role labeling. *arXiv preprint arXiv:1704.00405*, 2017. 3.5.2

[161] Meng Qu and Jian Tang. Probabilistic logic neural networks for reasoning. In *NeurIPS*, 2019. 2.1.2

[162] Meng Qu, Yoshua Bengio, and Jian Tang. Gmnn: Graph markov neural networks. In *ICML*, 2019. 2.1.2

[163] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/researchcovers/languageunsupervised/language understanding paper. pdf*, 2018. 4.4

[164] Martin Raison, Pierre-Emmanuel Mazaré, Rajarshi Das, and Antoine Bordes. Weaver: Deep co-encoding of questions and documents for machine reading. *arXiv preprint arXiv:1804.10490*, 2018. 2.1.3

[165] Nazneen Fatema Rajani, Bryan McCann, Caiming Xiong, and Richard Socher. Explain yourself! leveraging language models for commonsense reasoning. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4932–4942, 2019. 9.3

[166] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. 2016. 2.1.3, 3.1, 3.6, 4.1, 4.3.2, 9.1

[167] Lev Ratinov, Dan Roth, Doug Downey, and Mike Anderson. Local and global algorithms for disambiguation to wikipedia. In *ACL*, 2011. URL `http://cogcomp.org/papers/RRDA11.pdf`. 1

[168] Siva Reddy, Oscar Täckström, Michael Collins, Tom Kwiatkowski, Dipanjan Das, Mark Steedman, and Mirella Lapata. Transforming dependency structures to logical forms for semantic parsing. *Transactions of the Association for Computational Linguistics*, 4:127–140, 2016. 2.1.3

[169] Sebastian Riedel, Limin Yao, Andrew McCallum, and Benjamin M Marlin. Relation extraction with matrix factorization and universal schemas. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 74–84, 2013. 2.1.2, 2.1.3, 5.4.2

[170] Michael Ringgaard, Rahul Gupta, and Fernando CN Pereira. Sling: A framework for frame semantic parsing. *arXiv preprint arXiv:1710.07032*, 2017. 8.2.3

[171] Tim Rocktäschel and Sebastian Riedel. End-to-end differentiable proving. In *NeurIPS*, 2017. 2.1.2

[172] Pum-Mo Ryu, Myung-Gil Jang, and Hyun-Ki Kim. Open domain question answering using wikipedia-based knowledge model. *Information Processing and Management*, 50(5):683 – 692, 2014. ISSN 0306-4573. doi: https://doi.org/10.1016/j.ipm.2014.04.007. URL `http://www.sciencedirect.com/science/article/pii/S0306457314000351`. 2.1.3

[173] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1): 61–80, 2009. 2.2.2

[174] Jost Schatzmann, Blaise Thomson, Karl Weilhammer, Hui Ye, and Steve Young. Agenda-based user simulation for bootstrapping a pomdp dialogue system. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for*

Computational Linguistics; Companion Volume, Short Papers, pages 149–152. Association for Computational Linguistics, 2007. 6.5.1

[175] Jost Schatzmann, Blaise Thomson, and Steve Young. Statistical user simulation with a hidden agenda. *Proc SIGDial, Antwerp*, 273282(9), 2007. 6.1, 6.5.1

[176] Konrad Scheffler and Steve Young. Automatic learning of dialogue strategy using dialogue simulation and reinforcement learning. In *Proceedings of the second international conference on Human Language Technology Research*, pages 12–19. Morgan Kaufmann Publishers Inc., 2002. 6.1

[177] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. *arXiv preprint arXiv:1703.06103*, 2017. (document), 5.1, 5.3, 5.3

[178] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *ESWC*, 2018. 2.2.2

[179] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *ICLR*, 2017. 2.1.3, 3.4.1, 3.5.1, 7.2.2, 7.3.1

[180] Minjoon Seo, Sewon Min, Ali Farhadi, and Hannaneh Hajishirzi. Query-reduction networks for question answering. *ICLR*, 2017. 3.4.2, 3.4.2

[181] Minjoon Seo, Tom Kwiatkowski, Ankur P Parikh, Ali Farhadi, and Hannaneh Hajishirzi. Phrase-indexed question answering: A new challenge for scalable document comprehension. *EMNLP*, 2018. 1.1, 7.1, 7.3.2, 8.1

[182] Minjoon Seo, Jinhyuk Lee, Tom Kwiatkowski, Ankur P Parikh, Ali Farhadi, and Hannaneh Hajishirzi. Real-time open-domain question answering with dense-sparse phrase index. *ACL*, 2019. 7.2.3, 7.4, 8.1, 8.1, 8.2.1, 8.3.2

[183] Yelong Shen, Po-Sen Huang, Jianfeng Gao, and Weizhu Chen. Reasonet: Learning to stop reading in machine comprehension. *arXiv preprint arXiv:1609.05284*, 2016. 3.1, 3.5.1, 4.2.4

[184] Yelong Shen, Po-Sen Huang, Jianfeng Gao, and Weizhu Chen. Reasonet: Learning to stop reading in machine comprehension. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1047–1055. ACM, 2017. 2.1.3, 3.4.1

[185] Yelong Shen, Jianshu Chen, Po-Sen Huang, Yuqing Guo, and Jianfeng Gao. Reinforce-

walk: Learning to walk in graph with monte carlo tree search. In *NeurIPS*, 2018. 2.1.2

[186] Edward H Shortliffe and Bruce G Buchanan. A model of inexact reasoning in medicine. *Mathematical biosciences*, 23(3-4):351–379, 1975. 2.1.1

[187] Anshumali Shrivastava and Ping Li. Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips). In *Advances in Neural Information Processing Systems*, pages 2321–2329, 2014. 2.2.3, 8.1, 8.2.2

[188] Amit Singhal. Introducing the knowledge graph: things, not strings, May 2012. URL `https://googleblog.blogspot.com/2012/05/introducing-knowledge-graph-things-not.html`. 1

[189] Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. Reasoning with neural tensor networks for knowledge base completion. In *Advances in neural information processing systems*, pages 926–934, 2013. 7.1

[190] Alessandro Sordoni, Phillip Bachman, and Yoshua Bengio. Iterative alternating neural attention for machine reading. *arXiv preprint arXiv:1606.02245*, 2016. 3.1, 3.4.1, 3.5.1

[191] Amanda Spink, Dietmar Wolfram, Major BJ Jansen, and Tefko Saracevic. Searching the web: The public and their queries. *Journal of the Association for Information Science and Technology*, 52(3):226–234, 2001. 6.6

[192] Emma Strubell, Patrick Verga, Daniel Andor, David Weiss, and Andrew McCallum. Linguistically-informed self-attention for semantic role labeling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 5027–5038, 2018. 3.6

[193] Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. Probabilistic databases. *Synthesis lectures on data management*, 3(2):1–180, 2011. 9.3

[194] Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. End-to-end memory networks. In *Advances in Neural Information Processing Systems*, pages 2431–2439, 2015. 3.1, 3.3

[195] Haitian Sun*, Bhuwan Dhingra*, Manzil Zaheer, Kathryn Mazaitis, Ruslan Salakhutdinov, and William W. Cohen. Open domain question answering using early fusion of knowledge bases and text. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 4231–4242, 2018.

[196] Haitian Sun, Bhuwan Dhingra, Manzil Zaheer, Kathryn Mazaitis, Ruslan Salakhutdinov,

and William W Cohen. Open domain question answering using early fusion of knowledge bases and text. In *EMNLP*, 2018. 1.1, 8.3.1

[197] Haitian Sun, Tania Bedrax-Weiss, and William W Cohen. Pullnet: Open domain question answering with iterative retrieval on knowledge bases and text. In *EMNLP*, 2019. (document), 5.5, 8.3.1, 8.3.1, 8.1

[198] Swabha Swayamdipta. *Learning Algorithms for Broad-Coverage Semantic Parsing*. PhD thesis, Carnegie Mellon University Pittsburgh, PA, 2017. 3.5.2

[199] Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. *ACL*, 2015. 3.5.2

[200] A. Talmor and J. Berant. The web as a knowledge-base for answering complex questions. In *North American Association for Computational Linguistics (NAACL)*, 2018. 1.1, 5.1, 8.4

[201] Alon Talmor, Yanai Elazar, Yoav Goldberg, and Jonathan Berant. olmpics–on what language model pre-training captures. *arXiv preprint arXiv:1912.13283*, 2019. 9.3

[202] Ian Tenney, Patrick Xia, Berlin Chen, Alex Wang, Adam Poliak, R Thomas McCoy, Najoung Kim, Benjamin Van Durme, Samuel R Bowman, Dipanjan Das, et al. What do you learn from context? probing for sentence structure in contextualized word representations. *ICLR*, 2019. 7.3.4

[203] tf.RaggedTensors. *TensorFlow Ragged Tensors*, 2018. URL `https://www.tensorflow.org/guide/raggedtensors`. 8.2.2

[204] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016. URL `http://arxiv.org/abs/1605.02688`. 3.4.1

[205] Kristina Toutanova, Danqi Chen, Patrick Pantel, Hoifung Poon, Pallavi Choudhury, and Michael Gamon. Representing text for joint embedding of text and knowledge bases. In *EMNLP*, 2015. 2.1.2

[206] Trieu H Trinh and Quoc V Le. A simple method for commonsense reasoning. *arXiv preprint arXiv:1806.02847*, 2018. 9.3

[207] Adam Trischler, Zheng Ye, Xingdi Yuan, and Kaheer Suleman. Natural language comprehension with the epireader. *EMNLP*, 2016. 3.4.1, 3.5.1

[208] George Tsatsaronis, Georgios Balikas, Prodromos Malakasiotis, Ioannis Partalas, Matthias Zschunke, Michael R Alvers, Dirk Weissenborn, Anastasia Krithara, Sergios

Petridis, Dimitris Polychronopoulos, et al. An overview of the bioasq large-scale biomedical semantic indexing and question answering competition. *BMC bioinformatics*, 16(1): 138, 2015. 4.1

[209] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017. 2.2.1, 2.2.1, 4.4, 8.2.2

[210] Patrick Verga, David Belanger, Emma Strubell, Benjamin Roth, and Andrew McCallum. Multilingual relation extraction using compositional universal schema. *NAACL*, 2016. 2.1.2

[211] Ellen M Voorhees and Dawn M Tice. Building a question answering test collection. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 200–207, 2000. 2.1.3

[212] Denny Vrandečić and Markus Krötzsch. Wikidata: a free collaborative knowledge base. 2014. 2.1.2, 7.3.1, 8.2.3

[213] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International Conference on Machine Learning*, pages 1058–1066, 2013. 5.3

[214] Hai Wang, Takeshi Onishi, Kevin Gimpel, and David McAllester. Emergent logical structure in vector representations of neural readers. *2nd Workshop on Representation Learning for NLP, ACL*, 2017. 3.5.2

[215] Richard C Wang and William W Cohen. Language-independent set expansion of named entities using the web. In *Seventh IEEE international conference on data mining (ICDM 2007)*, pages 342–350. IEEE, 2007. 7.2.4

[216] Richard C Wang, Nico Schlaefer, William W Cohen, and Eric Nyberg. Automatic set expansion for list question answering. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 947–954. Association for Computational Linguistics, 2008. 7.2.4

[217] Shuohang Wang, Mo Yu, Jing Jiang, Wei Zhang, Xiaoxiao Guo, Shiyu Chang, Zhiguo Wang, Tim Klinger, Gerald Tesauro, and Murray Campbell. Evidence aggregation for answer re-ranking in open-domain question answering. *arXiv preprint arXiv:1711.05116*, 2017. 2.1.3

[218] Shuohang Wang, Mo Yu, Xiaoxiao Guo, Zhiguo Wang, Tim Klinger, Wei Zhang, Shiyu Chang, Gerald Tesauro, Bowen Zhou, and Jing Jiang. R^3: Reinforced reader-ranker for open-domain question answering. 2018. 2.1.3

[219] Shuohang Wang, Mo Yu, Xiaoxiao Guo, Zhiguo Wang, Tim Klinger, Wei Zhang, Shiyu Chang, Gerry Tesauro, Bowen Zhou, and Jing Jiang. R 3: Reinforced ranker-reader for open-domain question answering. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. 3.6

[220] Xiaoyan Wang, Pavan Kapanipathi, Ryan Musa, Mo Yu, Kartik Talamadupula, Ibrahim Abdelaziz, Maria Chang, Achille Fokoue, Bassem Makni, Nicholas Mattei, et al. Improving natural language inference using external knowledge in the science questions domain. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7208–7215, 2019. 5.5

[221] Yusuke Watanabe, Bhuwan Dhingra, and Ruslan Salakhutdinov. Question answering from unstructured text by retrieval and comprehension. *arXiv preprint arXiv:1703.08885*, 2017. (document), 1.1, 2.1.3, 5.3

[222] Johannes Welbl, Pontus Stenetorp, and Sebastian Riedel. Constructing datasets for multi-hop reading comprehension across documents. *arXiv preprint arXiv:1710.06481*, 2017. 3.4.2, 5.1

[223] Johannes Welbl, Pontus Stenetorp, and Sebastian Riedel. Constructing datasets for multi-hop reading comprehension across documents. *TACL*, 2018. (document), 3.1, 9.1

[224] Tsung-Hsien Wen, Milica Gašić, Nikola Mrkšić, Pei-Hao Su, David Vandyke, and Steve Young. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. *EMNLP*, 2015. 6.5.1

[225] Tsung-Hsien Wen, Milica Gašić, Nikola Mrkšić, Lina M. Rojas-Barahona, Pei-Hao Su, Stefan Ultes, David Vandyke, and Steve Young. A network-based end-to-end trainable task-oriented dialogue system. *arXiv preprint arXiv:1604.04562*, 2016. 6.1, 6.3, 6.3, 6.5.1

[226] Tsung-Hsien Wen, Milica Gašić, Nikola Mrkšić, Lina M. Rojas-Barahona, Pei-Hao Su, Stefan Ultes, David Vandyke, and Steve Young. Conditional generation and snapshot learning in neural dialogue systems. *EMNLP*, 2016. 6.5.1

[227] Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M Rush, Bart van Merriënboer, Armand Joulin, and Tomas Mikolov. Towards ai-complete question answering: A set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*, 2015. (document), 3.1, 3.4.2, 9.1

[228] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *ICLR*, 2015. 3.1, 3.4.1, 3.5.1

[229] Georg Wiese, Dirk Weissenborn, and Mariana Neves. Neural domain adaptation for biomedical question answering. In *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017)*, pages 281–289, Vancouver, Canada, August 2017. Association for Computational Linguistics. URL http://aclweb.org/anthology/K17-1029. 5.1

[230] Georg Wiese, Dirk Weissenborn, and Mariana L. Neves. Neural question answering at bioasq 5b. In *BioNLP 2017, Vancouver, Canada, August 4, 2017*, pages 76–79, 2017. doi: 10.18653/v1/W17-2309. URL https://doi.org/10.18653/v1/W17-2309. (document), 4.3.2, 4.4, 4.3.2

[231] Jason D Williams and Steve Young. Scaling up POMDPs for dialog management: The "Summary POMDP" method. In *IEEE Workshop on Automatic Speech Recognition and Understanding, 2005.*, pages 177–182. IEEE, 2005. 6.3

[232] Jason D Williams and Geoffrey Zweig. End-to-end lstm-based dialog control optimized with supervised and reinforcement learning. *arXiv preprint arXiv:1606.01269*, 2016. 6.3, 6.6

[233] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992. 6.4, 6.4

[234] Terry Winograd. Procedures as a representation for data in a computer program for understanding natural language. Technical report, MASSACHUSETTS INST OF TECH CAMBRIDGE PROJECT MAC, 1971. 2.1.1

[235] Sam Wiseman, Alexander M Rush, and Stuart M Shieber. Learning global features for coreference resolution. *NAACL*, 2016. 3.1

[236] Ji Wu, Miao Li, and Chin-Hui Lee. A probabilistic framework for representing dialog systems and entropy-based dialog management through dynamic stochastic state evolution. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(11):2026–2035, 2015. 6.3, 6.5.2, 6.6

[237] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016. 4.4, 7.2.2

[238] Yuhuai Wu, Saizheng Zhang, Ying Zhang, Yoshua Bengio, and Ruslan Salakhutdinov. On multiplicative integration with recurrent neural networks. *Advances in Neural Information Processing Systems*, 2016. 3.2

[239] Wenhan Xiong, Thien Hoang, and William Yang Wang. Deeppath: A reinforcement learning method for knowledge graph reasoning. In *EMNLP*, 2017. 2.1.2

[240] Bishan Yang and Tom Mitchell. Leveraging knowledge bases in lstms for improving machine reading. 2017. 5.5

[241] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint arXiv:1412.6575*, 2014. 8.4

[242] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Learning multi-relational semantics using neural-embedding models. *NIPS Workshop on Learning Semantics*, 2014. 3.2

[243] Zhilin Yang, Ruslan Salakhutdinov, and William Cohen. Multi-task cross-lingual sequence tagging from scratch. *arXiv preprint arXiv:1603.06270*, 2016. 3.2

[244] Zhilin Yang, Bhuwan Dhingra, Ye Yuan, Junjie Hu, William W Cohen, and Ruslan Salakhutdinov. Words or characters? fine-grained gating for reading comprehension. *ICLR*, 2017. 3.6

[245] Zhilin Yang, Junjie Hu, Ruslan Salakhutdinov, and William W Cohen. Semi-supervised qa with generative domain-adaptive nets. *ACL*, 2017. 4.1, 4.3.2, 2, 4.3.2

[246] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *EMNLP*, 2018. (document), 1.1, 8.1, 8.4, 8.3.3, 8.3.3, 8.3.4, 8.3.3, 9.1

[247] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems*, pages 5754–5764, 2019. 1.1

[248] Zichao Yang, Phil Blunsom, Chris Dyer, and Wang Ling. Reference-aware language models. *EMNLP*, 2017. 3.5.2

[249] Kaisheng Yao, Baolin Peng, Yu Zhang, Dong Yu, Geoffrey Zweig, and Yangyang Shi. Spoken language understanding using long short-term memory neural networks. In

Spoken Language Technology Workshop (SLT), 2014 IEEE, pages 189–194. IEEE, 2014. 6.3

[250] Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1321–1331, Beijing, China, July 2015. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P15-1128`. 1

[251] Wen-tau Yih, Matthew Richardson, Chris Meek, Ming-Wei Chang, and Jina Suh. The value of semantic parse labeling for knowledge base question answering. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 201–206, 2016. 5.1, 5.4.1

[252] Jun Yin, Xin Jiang, Zhengdong Lu, Lifeng Shang, Hang Li, and Xiaoming Li. Neural generative question answering. *International Joint Conference on Artificial Intelligence*, 2016. 6.6

[253] Pengcheng Yin, Zhengdong Lu, Hang Li, and Ben Kao. Neural enquirer: Learning to query tables. *International Joint Conference on Artificial Intelligence*, 2016. 6.6

[254] Steve Young, Milica Gašić, Blaise Thomson, and Jason D Williams. Pomdp-based statistical spoken dialog systems: A review. *Proceedings of the IEEE*, 101(5):1160–1179, 2013. 6.1, 6.3

[255] Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V Le. Qanet: Combining local convolution with global self-attention for reading comprehension. *ICLR*, 2018. 2.1.3

[256] Mo Yu, Wenpeng Yin, Kazi Saidul Hasan, Cicero dos Santos, Bing Xiang, and Bowen Zhou. Improved neural relation detection for knowledge base question answering. *ACL*, 2017. 5.4.2

[257] Wojciech Zaremba and Ilya Sutskever. Reinforcement learning neural Turing machines-revised. *arXiv preprint arXiv:1505.00521*, 2015. 6.6

[258] John M Zelle and Raymond J Mooney. Learning to parse database queries using inductive logic programming. In *Proceedings of the national conference on artificial intelligence*, 1996. 2.1.3

[259] Luke S Zettlemoyer and Michael Collins. Learning to map sentences to logical form:

Structured classification with probabilistic categorial grammars. In *UAI*, 2005. 2.1.3

[260] Yuyu Zhang, Hanjun Dai, Zornitsa Kozareva, Alexander J Smola, and Le Song. Variational reasoning for question answering with knowledge graph. In *AAAI*, 2018. 8.3.1, 9.1

[261] Tiancheng Zhao and Maxine Eskenazi. Towards end-to-end learning for dialog state tracking and management using deep reinforcement learning. *arXiv preprint arXiv:1606.02560*, 2016. 6.3, 6.6

[262] Stefan Zwicklbauer, Christin Seifert, and Michael Granitzer. Do we need entity-centric knowledge bases for entity disambiguation? In *Proceedings of the 13th International Conference on Knowledge Management and Knowledge Technologies*, page 4. ACM, 2013. 6.2