CHAPTER

28 Question Answering

The quest for knowledge is deeply human, and so it is not surprising that practically as soon as there were computers, and certainly as soon as there was natural language processing, we were trying to use computers to answer textual questions. By the early 1960s, there were systems implementing the two major modern paradigms of question answering—**IR-based question answering** and **knowledge-based question answering** to answer questions about baseball statistics or scientific facts. Even imaginary computers got into the act. Deep Thought, the computer that Douglas Adams invented in *The Hitchhiker's Guide to the Galaxy*, managed to answer "the Great Question Of Life The Universe and Everything" (the answer was 42, but unfortunately the details of the question were never revealed).

More recently, IBM's Watson question-answering system won the TV gameshow *Jeopardy!* in 2011, beating humans at the task of answering questions like

> WILLIAM WILKINSON'S "AN ACCOUNT OF THE PRINCIPALITIES OF WALLACHIA AND MOLDOVIA" INSPIRED THIS AUTHOR'S MOST FAMOUS NOVEL[1]

Although the goal of quiz shows is entertainment, the technology used to answer these questions both draws on and extends the state of the art in practical question answering, as we will see.

Most current question answering systems focus on **factoid questions**. Factoid questions are questions that can be answered with simple facts expressed in short text answers. The following factoid questions, for example, can be answered with a short string expressing a personal name, temporal expression, or location:

(28.1) Who founded Virgin Airlines?

(28.2) What is the average age of the onset of autism?

(28.3) Where is Apple Computer based?

In this chapter we describe the two major modern paradigms to question answering, focusing on their application to factoid questions.

The first paradigm is called **IR-based question answering** or sometimes **text-based question answering**, and relies on the enormous amounts of information available as text on the Web or in specialized collections such as PubMed. Given a user question, information retrieval techniques extract passages directly from these documents, guided by the text of the question.

The method processes the question to determine the likely **answer type** (often a named entity like a person, location, or time), and formulates queries to send to a search engine. The search engine returns ranked documents which are broken up into suitable passages and reranked. Finally candidate answer strings are extracted from the passages and ranked.

[1] The answer, of course, is Bram Stoker, and the novel was the fantastically Gothic *Dracula*.

In the second paradigm, **knowledge-based question answering**, we instead build a semantic representation of the query. The meaning of a query can be a full predicate calculus statement. So the question *What states border Texas?*—taken from the GeoQuery database of questions on U.S. Geography (Zelle and Mooney, 1996)— might have the representation:

$\lambda x.state(x) \wedge borders(x, texas)$

Alternatively the meaning of a question could be a single relation between a known and an unknown entity. Thus the representation of the question *When was Ada Lovelace born?* could be `birth-year (Ada Lovelace, ?x)`.

Whatever meaning representation we choose, we'll be using it to query databases of facts. These might be complex databases, perhaps of scientific facts or geospatial information, that need powerful logical or SQL queries. Or these might be databases of simple relations, **triple stores** like Freebase or DBpedia introduced in Chapter 20.

triple stores

Large practical systems like the DeepQA system in IBM's Watson generally are hybrid systems, using both text datasets and structured knowledge bases to answer questions. DeepQA extracts a wide variety of meanings from the question (parses, relations, named entities, ontological information), and then finds large numbers of candidate answers in both knowledge bases and in textual sources like Wikipedia or newspapers. Each candidate answer is then scored using a wide variety of knowledge sources, such as geospatial databases, temporal reasoning, taxonomical classification, and various textual sources.

We'll explore all three of these approaches: IR-based, knowledge-based, and the Watson DeepQA system, in the next three sections.

28.1 IR-based Factoid Question Answering

The goal of IR-based question answering is to answer a user's question by finding short text segments on the Web or some other collection of documents. Figure 28.1 shows some sample factoid questions and their answers.

Question	Answer
Where is the Louvre Museum located?	in Paris, France
What's the abbreviation for limited partnership?	L.P.
What are the names of Odin's ravens?	Huginn and Muninn
What currency is used in China?	the yuan
What kind of nuts are used in marzipan?	almonds
What instrument does Max Roach play?	drums
What's the official language of Algeria?	Arabic
How many pounds are there in a stone?	14

Figure 28.1 Some sample factoid questions and their answers.

Figure 28.2 shows the three phases of an IR-based factoid question-answering system: question processing, passage retrieval and ranking, and answer processing.

28.1.1 Question Processing

The goal of the question-processing phase is to extract a number of pieces of information from the question. The **answer type** specifies the kind of entity the answer consists of (person, location, time, etc.). The **query** specifies the keywords that should be used for the IR system to use in searching for documents. Some systems

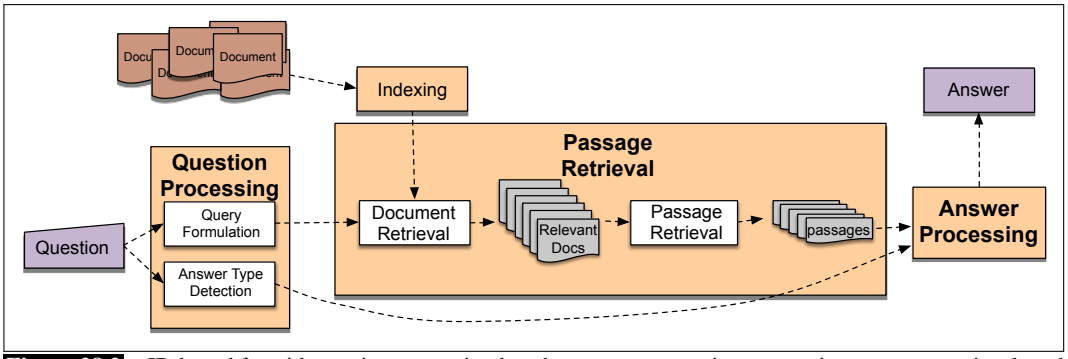


Figure 28.2 IR-based factoid question answering has three stages: question processing, passage retrieval, and answer processing.

also extract a **focus**, which is the string of words in the question that are likely to be replaced by the answer in any answer string found. Some systems also classify the **question type**: is this a definition question, a math question, a list question? For example, for the following question:

Which US state capital has the largest population?

The query processing should produce results like the following:

Answer Type: city

Query: US state capital, largest, population

Focus: state capital

In the next two sections we summarize the two most commonly used tasks, answer type detection and query formulation.

28.1.2 Answer Type Detection (Question Classification)

question classification
answer type

The task of **question classification** or **answer type recognition** is to determine the **answer type**, the named-entity or similar class categorizing the answer. A question like *"Who founded Virgin Airlines"* expects an answer of type PERSON. A question like *"What Canadian city has the largest population?"* expects an answer of type CITY. If we know the answer type for a question, we can avoid looking at every sentence or noun phrase in the entire suite of documents for the answer, instead focusing on, for example, just people or cities.

As some of the above examples suggest, we might draw the set of possible answer types for a question classifier from a set of named entities like PERSON, LOCATION, and ORGANIZATION described in Chapter 20. Usually, however, a richer, often hierarchical set of answer types is used, an **answer type taxonomy**. Such taxonomies can be built semi-automatically and dynamically, for example, from WordNet (Harabagiu et al. 2000,Pasca 2003), or they can be designed by hand.

answer type taxonomy

Figure 28.4 shows one such hand-built ontology, the Li and Roth (2005) tagset; a subset is shown graphically in Fig. 28.3. In this hierarchical tagset, each question can be labeled with a coarse-grained tag like HUMAN or a fine-grained tag like HUMAN:DESCRIPTION, HUMAN:GROUP, HUMAN:IND, and so on. Similar tags are used in other systems; the HUMAN:DESCRIPTION type is often called a BIOGRAPHY question because the answer is required to give a brief biography of the person rather than just a name.

Question classifiers can be built by hand-writing rules, by supervised machine learning, or with some combination. The Webclopedia QA Typology, for example,

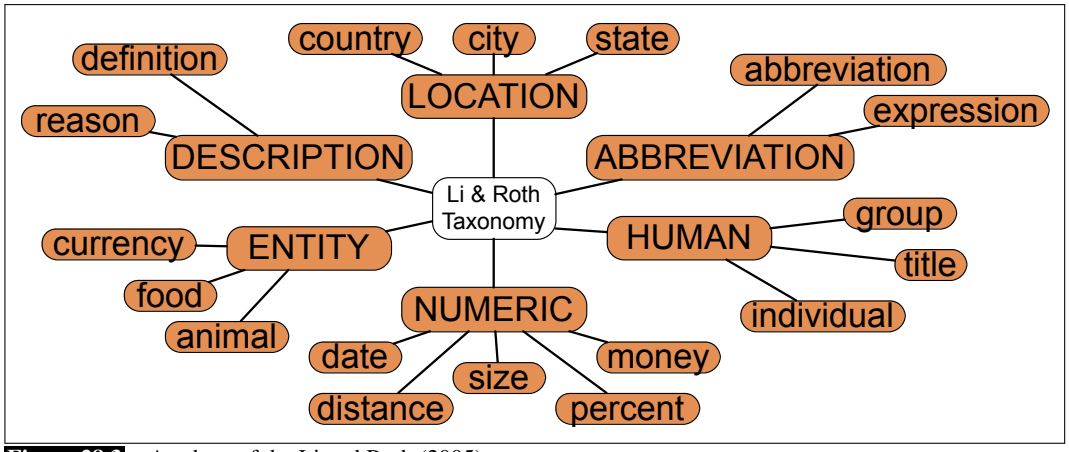


Figure 28.3 A subset of the Li and Roth (2005) answer types.

contains 276 hand-written rules associated with the approximately 180 answer types in the typology (Hovy et al., 2002). A regular expression rule for detecting an answer type like BIOGRAPHY (which assumes the question has been named-entity-tagged) might be

(28.4) who {is | was | are | were} PERSON

Most modern question classifiers, however, are based on supervised machine learning, and are trained on databases of questions that have been hand-labeled with an answer type (Li and Roth, 2002). Typical features used for classification include the words in the questions, the part-of-speech of each word, and named entities in the questions.

Often, a single word in the question gives extra information about the answer type, and its identity is used as a feature. This word is sometimes called the **answer type word** or **question headword**, and may be defined as the headword of the first NP after the question's *wh-word*; headwords are indicated in boldface in the following examples:

(28.5) Which **city** in China has the largest number of foreign financial companies?

(28.6) What is the state **flower** of California?

Finally, it often helps to use semantic information about the words in the questions. The WordNet synset ID of the word can be used as a feature, as can the IDs of the hypernym and hyponyms of each word in the question.

In general, question classification accuracies are relatively high on easy question types like PERSON, LOCATION, and TIME questions; detecting REASON and DESCRIPTION questions can be much harder.

28.1.3 Query Formulation

Query formulation is the task of creating from the question a list of keywords that form a query that can be sent to an information retrieval system. Exactly what query to form depends on the application. If question answering is applied to the Web, we might simply create a keyword from every word in the question, letting the Web search engine automatically remove any stopwords. Often, we leave out the question word (*where*, *when*, etc.). Alternatively, keywords can be formed from only the terms found in the noun phrases in the question, applying stopword lists to ignore function words and high-frequency, low-content verbs.

Tag	Example
ABBREVIATION	
abb	What's the abbreviation for limited partnership?
exp	What does the "c" stand for in the equation E=mc2?
DESCRIPTION	
definition	What are tannins?
description	What are the words to the Canadian National anthem?
manner	How can you get rust stains out of clothing?
reason	What caused the Titanic to sink ?
ENTITY	
animal	What are the names of Odin's ravens?
body	What part of your body contains the corpus callosum?
color	What colors make up a rainbow ?
creative	In what book can I find the story of Aladdin?
currency	What currency is used in China?
disease/medicine	What does Salk vaccine prevent?
event	What war involved the battle of Chapultepec?
food	What kind of nuts are used in marzipan?
instrument	What instrument does Max Roach play?
lang	What's the official language of Algeria?
letter	What letter appears on the cold-water tap in Spain?
other	What is the name of King Arthur's sword?
plant	What are some fragrant white climbing roses?
product	What is the fastest computer?
religion	What religion has the most members?
sport	What was the name of the ball game played by the Mayans?
substance	What fuel do airplanes use?
symbol	What is the chemical symbol for nitrogen?
technique	What is the best way to remove wallpaper?
term	How do you say " Grandma " in Irish?
vehicle	What was the name of Captain Bligh's ship?
word	What's the singular of dice?
HUMAN	
description	Who was Confucius?
group	What are the major companies that are part of Dow Jones?
ind	Who was the first Russian astronaut to do a spacewalk?
title	What was Queen Victoria's title regarding India?
LOCATION	
city	What's the oldest capital city in the Americas?
country	What country borders the most others?
mountain	What is the highest peak in Africa?
other	What river runs through Liverpool?
state	What states do not have state income tax?
NUMERIC	
code	What is the telephone number for the University of Colorado?
count	About how many soldiers died in World War II?
date	What is the date of Boxing Day?
distance	How long was Mao's 1930s Long March?
money	How much did a McDonald's hamburger cost in 1963?
order	Where does Shanghai rank among world cities in population?
other	What is the population of Mexico?
period	What was the average life expectancy during the Stone Age?
percent	What fraction of a beaver's life is spent swimming?
temp	How hot should the oven be when making Peachy Oat Muffins?
speed	How fast must a spacecraft travel to escape Earth's gravity?
size	What is the size of Argentina?
weight	How many pounds are there in a stone?

Figure 28.4 Question typology from Li and Roth (2002), (2005). Example sentences are from their corpus of 5500 labeled questions. A question can be labeled either with a coarse-grained tag like HUMAN or NUMERIC or with a fine-grained tag like HUMAN:DESCRIPTION, HUMAN:GROUP, HUMAN:IND, and so on.

When question answering is applied to smaller sets of documents, for example, to answer questions about corporate information pages, we still use an IR engine to search our documents for us. But for this smaller set of documents, we generally need to apply query expansion. On the Web the answer to a question might appear in many different forms, so if we search with words from the question we'll probably find an answer written in the same form. In smaller sets of corporate pages, by contrast, an answer might appear only once, and the exact wording might look nothing like the question. Thus, query expansion methods can add query terms in hopes of matching the particular form of the answer as it appears. These might include all morphological variants of the content words in the question, or synonyms from a thesaurus.

query reformulation

A query formulation approach that is sometimes used for questioning the Web is to apply **query reformulation** rules to the query. The rules rephrase the question to make it look like a substring of possible declarative answers. The question *"when was the laser invented?"* might be reformulated as *"the laser was invented"*; the question *"where is the Valley of the Kings?"* as *"the Valley of the Kings is located in"*. Here are some sample hand-written reformulation rules from Lin (2007):

(28.7) *wh-word* did A *verb* B → ... A *verb*+ed B

(28.8) Where is A → A is located in

28.1.4 Passage Retrieval

The query that was created in the question-processing phase is next used to query an information-retrieval system, either a general IR engine over a proprietary set of indexed documents or a Web search engine. The result of this document retrieval stage is a set of documents.

Although the set of documents is generally ranked by relevance, the top-ranked document is probably not the answer to the question. This is because documents are not an appropriate unit to rank with respect to the goals of a question-answering system. A highly relevant and large document that does not prominently answer a question is not an ideal candidate for further processing.

Therefore, the next stage is to extract a set of potential answer passages from the retrieved set of documents. The definition of a passage is necessarily system dependent, but the typical units include sections, paragraphs, and sentences. We might run a paragraph segmentation algorithm on all the returned documents and treat each paragraph as a segment.

passage retrieval

We next perform **passage retrieval**. In this stage, we first filter out passages in the returned documents that don't contain potential answers and then rank the rest according to how likely they are to contain an answer to the question. The first step in this process is to run a named entity or answer type classification on the retrieved passages. The answer type that we determined from the question tells us the possible answer types we expect to see in the answer. We can therefore filter out documents that don't contain any entities of the right type.

The remaining passages are then ranked, usually by supervised machine learning, relying on a small set of features that can be easily extracted from a potentially large number of answer passages, such as:

- The number of **named entities** of the right type in the passage
- The number of **question keywords** in the passage
- The longest exact sequence of question keywords that occurs in the passage
- The rank of the document from which the passage was extracted

- The **proximity** of the keywords from the original query to each other
  For each passage identify the shortest span that covers the keywords contained in that passage. Prefer smaller spans that include more keywords (Pasca 2003, Monz 2004).
- The ***N*-gram overlap** between the passage and the question
  Count the *N*-grams in the question and the *N*-grams in the answer passages. Prefer the passages with higher *N*-gram overlap with the question (Brill et al., 2002).

For question answering from the Web, instead of extracting passages from all returned documents, we can rely on the Web search to do passage extraction for us. We do this by using **snippets** produced by the Web search engine as the returned passages. For example, Fig. 28.5 shows snippets for the first five documents returned from Google for the query *When was movable type metal printing invented in Korea?*

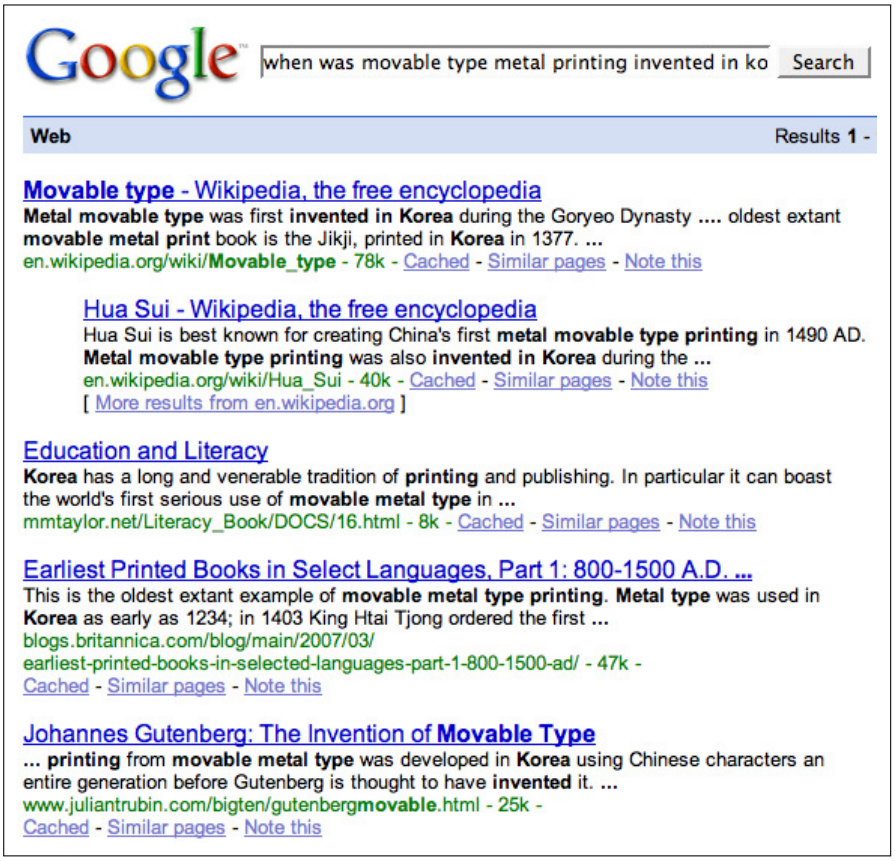


Figure 28.5 Five snippets from Google in response to the query *When was movable type metal printing invented in Korea?*

28.1.5 Answer Processing

The final stage of question answering is to extract a specific answer from the passage so as to be able to present the user with an answer like *29,029 feet* to the question *"How tall is Mt. Everest?"*

Two classes of algorithms have been applied to the answer-extraction task, one based on **answer-type pattern extraction** and one based on **N-gram tiling**.

In the **pattern-extraction** methods for answer processing, we use information about the expected answer type together with regular expression patterns. For example, for questions with a HUMAN answer type, we run the answer type or named entity tagger on the candidate passage or sentence and return whatever entity is labeled with type HUMAN. Thus, in the following examples, the underlined named entities are extracted from the candidate answer passages as the answer to the HUMAN and DISTANCE-QUANTITY questions:

"Who is the prime minister of India"
Manmohan Singh, Prime Minister of India, had told left leaders that the deal would not be renegotiated.

"How tall is Mt. Everest?"
The official height of Mount Everest is 29029 feet

Unfortunately, the answers to some questions, such as DEFINITION questions, don't tend to be of a particular named entity type. For some questions, then, instead of using answer types, we use hand-written regular expression patterns to help extract the answer. These patterns are also useful in cases in which a passage contains multiple examples of the same named entity type. Figure 28.6 shows some patterns from Pasca (2003) for the question phrase (QP) and answer phrase (AP) of definition questions.

Pattern	Question	Answer
<AP> such as <QP>	What is autism?	", developmental disorders such as autism"
<QP>, a <AP>	What is a caldera?	"the Long Valley caldera, a volcanic crater 19 miles long"

Figure 28.6 Some answer-extraction patterns for definition questions (Pasca, 2003).

The patterns are specific to each question type and can either be written by hand or learned automatically using relation extraction methods. Patterns can then be used together with other information as features in a classifier that ranks candidate answers. We extract potential answers by using named entities or patterns or even just by looking at every sentence returned from passage retrieval and rank them using a classifier with features like the following.

Answer type match: True if the candidate answer contains a phrase with the correct answer type.

Pattern match: The identity of a pattern that matches the candidate answer.

Number of matched question keywords: How many question keywords are contained in the candidate answer.

Keyword distance: The distance between the candidate answer and query keywords (measured in average number of words or as the number of keywords that occur in the same syntactic phrase as the candidate answer).

Novelty factor: True if at least one word in the candidate answer is novel, that is, not in the query.

Apposition features: True if the candidate answer is an appositive to a phrase containing many question terms. Can be approximated by the number of question terms separated from the candidate answer through at most three words and one comma (Pasca, 2003).

Punctuation location: True if the candidate answer is immediately followed by a comma, period, quotation marks, semicolon, or exclamation mark.

Sequences of question terms: The length of the longest sequence of question terms that occurs in the candidate answer.

N-gram tiling

An alternative approach to answer extraction, used solely in Web search, is based on **N-gram tiling**, sometimes called the **redundancy-based approach** (Brill et al. 2002, Lin 2007). This simplified method begins with the snippets returned from the Web search engine, produced by a reformulated query. In the first step,

N-gram mining

N-gram mining, every unigram, bigram, and trigram occurring in the snippet is extracted and weighted. The weight is a function of the number of snippets in which the N-gram occurred, and the weight of the query reformulation pattern that returned it. In the **N-gram filtering** step, N-grams are scored by how well they match the predicted answer type. These scores are computed by hand-written filters built for each answer type. Finally, an **N-gram tiling** algorithm concatenates overlapping N-gram fragments into longer answers. A standard greedy method is to start with the highest-scoring candidate and try to tile each other candidate with this candidate. The best-scoring concatenation is added to the set of candidates, the lower-scoring candidate is removed, and the process continues until a single answer is built.

N-gram filtering

For any of these answer-extraction methods, the exact answer phrase can just be presented to the user by itself, or, more helpfully, accompanied by enough passage information to provide helpful context.

28.2 Knowledge-based Question Answering

While an enormous amount of information is encoded in the vast amount of text on the web, information obviously also exists in more structured forms. We use the term **knowledge-based question answering** for the idea of answering a natural language question by mapping it to a query over a structured database. Like the text-based paradigm for question answering, this approach dates back to the earliest days of natural language processing, with systems like BASEBALL (Green et al., 1961) that answered questions from a structured database of baseball games and stats.

Systems for mapping from a text string to any logical form are called **semantic parsers** (???). Semantic parsers for question answering usually map either to some version of predicate calculus or a query language like SQL or SPARQL, as in the examples in Fig. 28.7.

Question	Logical form
When was Ada Lovelace born?	`birth-year (Ada Lovelace, ?x)`
What states border Texas?	λ x.state(x) $\wedge$ borders(x,texas)
What is the largest state	argmax(λx.state(x), λx.size(x))
How many people survived the sinking of the Titanic	`(count (!fb:event.disaster.survivors fb:en.sinking_of_the_titanic))`

Figure 28.7 Sample logical forms produced by a semantic parser for question answering. These range from simple relations like `birth-year`, or relations normalized to databases like Freebase, to full predicate calculus.

The logical form of the question is thus either in the form of a query or can easily be converted into one. The database can be a full relational database, or simpler structured databases like sets of **RDF triples**. Recall from Chapter 20 that an RDF triple is a 3-tuple, a predicate with two arguments, expressing some simple relation

or proposition. Popular ontologies like Freebase (Bollacker et al., 2008) or DBpedia (Bizer et al., 2009) have large numbers of triples derived from Wikipedia **infoboxes**, the structured tables associated with certain Wikipedia articles.

The simplest formation of the knowledge-based question answering task is to answer factoid questions that ask about one of the missing arguments in a triple. Consider an RDF triple like the following:

subject	**predicate**	**object**
Ada Lovelace	birth-year	1815

This triple can be used to answer text questions like 'When was Ada Lovelace born?' or 'Who was born in 1815?'. Question answering in this paradigm requires mapping from textual strings like "When was ... born" to canonical relations in the knowledge base like *birth-year*. We might sketch this task as:

"When was Ada Lovelace born?" → `birth-year (Ada Lovelace, ?x)`
"What is the capital of England?" → `capital-city(?x, England)`

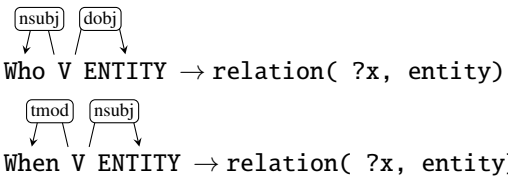
28.2.1 Rule-based Methods

For relations that are very frequent, it may be worthwhile to write hand-written rules to extract relations from the question, just as we saw in Section **??**. For example, to extract the birth-year relation, we could write patterns that search for the question word *When*, a main verb like *born*, and that extract the named entity argument of the verb.

28.2.2 Supervised Methods

In some cases we have supervised data, consisting of a set of questions paired with their correct logical form like the examples in Fig. 28.7. The task is then to take those pairs of training tuples and produce a system that maps from new questions to their logical forms.

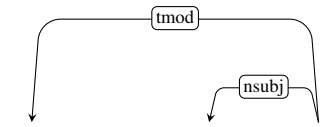
Most supervised algorithms for learning to answer these simple questions about relations first parse the questions and then align the parse trees to the logical form. Generally these systems bootstrap by having a small set of rules for building this mapping, and an initial lexicon as well. For example, a system might have built-in strings for each of the entities in the system (`Texas`, `Ada Lovelace`), and then have simple default rules mapping fragments of the question parse tree to particular relations:



Then given these rules and the lexicon, a training tuple like the following:

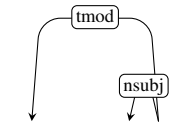
"When was Ada Lovelace born?" → `birth-year (Ada Lovelace, ?x)`

would first be parsed, resulting in the following mapping.



When was Ada Lovelace born → birth-year(Ada Lovelace, ?x)

From many pairs like this, we could induce mappings between pieces of parse fragment, such as the mapping between the parse fragment on the left and the relation on the right:



When was · born → birth-year(, ?x)

A supervised system would thus parse each tuple in the training set and induce a bigger set of such specific rules, allowing it to map unseen examples of "When was X born?" questions to the `birth-year` relation. Rules can furthermore be associated with counts based on the number of times the rule is used to parse the training data. Like rule counts for probabilistic grammars, these can be normalized into probabilities. The probabilities can then be used to choose the highest probability parse for sentences with multiple semantic interpretations.

The supervised approach can be extended to deal with more complex questions that are not just about single relations. Consider the question *What is the biggest state bordering Texas?* from the GEOQUERY (Zelle and Mooney, 1996) dataset, with the semantic form:

$\text{argmax}(\lambda x.state(x) \wedge borders(x, texas), \lambda x.size(x))$

This question has much more complex structures than the simple single-relation questions we considered above, such as the argmax function, the mapping of the word *biggest* to *size* and so on. Zettlemoyer and Collins (2005) shows how more complex default rules (along with richer syntactic structures) can be used to learn to map from text sentences to more complex logical forms. The rules take the training set's pairings of sentence and meaning as above and use the complex rules to break each training example down into smaller tuples that can then be recombined to parse new sentences.

28.2.3 Dealing with Variation: Semi-Supervised Methods

Because it is difficult to create training sets with questions labeled with their meaning representation, supervised datasets can't cover the wide variety of forms that even simple factoid questions can take. For this reason most techniques for mapping factoid questions to the canonical relations or other structures in knowledge bases find some way to make use of textual redundancy.

The most common source of redundancy, of course, is the web, which contains vast number of textual variants expressing any relation. For this reason, most methods make some use of web text, either via semi-supervised methods like **distant supervision** or unsupervised methods like **open information extraction**, both introduced in Chapter 20. For example the REVERB open information extractor (Fader et al., 2011) extracts billions of (subject, relation, object) triples of strings from the web, such as ("Ada Lovelace","was born in", "1815"). By **aligning** these strings with a canonical knowledge source like Wikipedia, we create new relations that can be queried while simultaneously learning to map between the words in question and

canonical relations.

To align a REVERB triple with a canonical knowledge source we first align the arguments and then the predicate. Recall from Chapter 23 that linking a string like "Ada Lovelace" with a Wikipedia page is called **entity linking**; we thus represent the concept 'Ada Lovelace' by a unique identifier of a Wikipedia page. If this subject string is not associated with a unique page on Wikipedia, we can disambiguate which page is being sought, for example by using the cosine distance between the triple string ('Ada Lovelace was born in 1815') and each candidate Wikipedia page. Date strings like '1815' can be turned into a normalized form using standard tools for temporal normalization like SUTime (Chang and Manning, 2012). Once we've aligned the arguments, we align the predicates. Given the Freebase relation `people.person.birthdate(ada lovelace,1815)` and the string 'Ada Lovelace was born in 1815', having linked Ada Lovelace and normalized 1815, we learn the mapping between the string 'was born in' and the relation people.person.birthdate. In the simplest case, this can be done by aligning the relation with the string of words in between the arguments; more complex alignment algorithms like IBM Model 1 (Chapter 25) can be used. Then if a phrase aligns with a predicate across many entities, it can be extracted into a lexicon for mapping questions to relations.

entity linking

Here are some examples from such a resulting lexicon, produced by Berant et al. (2013), giving many variants of phrases that align with the Freebase relation `country.capital` between a country and its capital city:

capital of	capital city of	become capital of
capitol of	national capital of	official capital of
political capital of	administrative capital of	beautiful capital of
capitol city of	remain capital of	make capital of
political center of	bustling capital of	capital city in
cosmopolitan capital of	move its capital to	modern capital of
federal capital of	beautiful capital city of	administrative capital city of

Figure 28.8 Some phrases that align with the Freebase relation `country.capital` from Berant et al. (2013).

Another useful source of linguistic redundancy are paraphrase databases. For example the site `wikianswers.com` contains millions of pairs of questions that users have tagged as having the same meaning, 18 million of which have been collected in the PARALEX corpus (Fader et al., 2013). Here's an example:

Q: What are the green blobs in plant cells?

Lemmatized synonyms from PARALEX:

what be the green blob in plant cell?
what be green part in plant cell?
what be the green part of a plant cell?
what be the green substance in plant cell?
what be the part of plant cell that give it green color?
what cell part do plant have that enable the plant to be give a green color?
what part of the plant cell turn it green?
part of the plant cell where the cell get it green color?
the green part in a plant be call?
the part of the plant cell that make the plant green be call?

The resulting millions of pairs of question paraphrases can be aligned to each other using MT alignment approaches (such as IBM Model 1) to create an MT-style

phrase table for translating from question phrases to synonymous phrases. These are used by a number of modern question answering algorithms, generating all paraphrases of a question as part of the process of finding an answer (Fader et al. 2013, Berant and Liang 2014).

28.3 Using multiple information sources: IBM's Watson

Of course there is no reason to limit ourselves to just text-based or knowledge-based resources for question answering. The Watson system from IBM that won the Jeopardy! challenge in 2011 is an example of a system that relies on a wide variety of resources to answer questions.

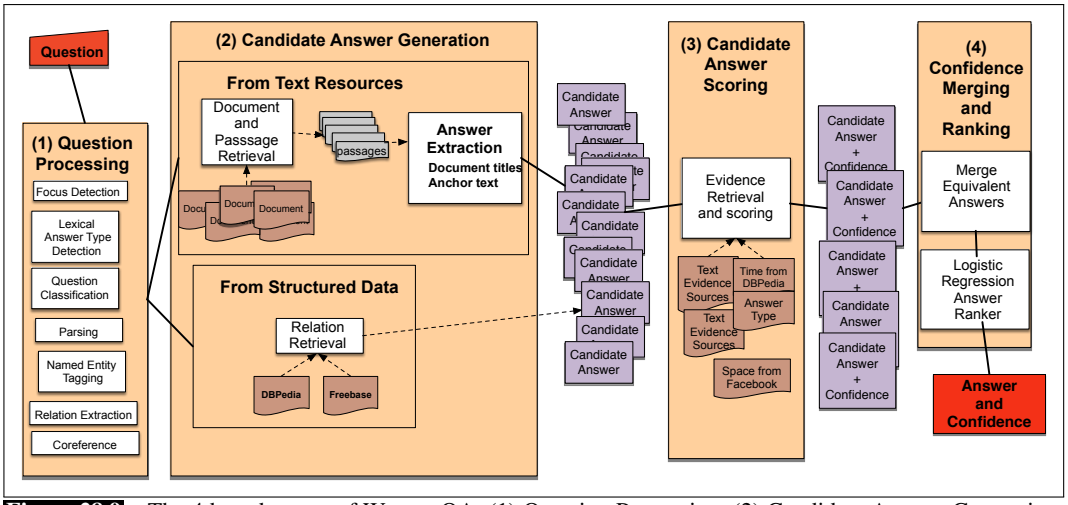


Figure 28.9 The 4 broad stages of Watson QA: (1) Question Processing, (2) Candidate Answer Generation, (3) Candidate Answer Scoring, and (4) Answer Merging and Confidence Scoring.

Figure 28.9 shows the 4 stages of the DeepQA system that is the question answering component of Watson.

The first stage is **question processing**. The DeepQA system runs parsing, named entity tagging, and relation extraction on the question. Then, like the text-based systems in Section 28.1, the DeepQA system extracts the **focus**, the **answer type** (also called the **lexical answer type** or **LAT**), and performs **question classification** and **question sectioning**.

Consider these Jeopardy! examples, with a category followed by a question:

> Poets and Poetry: **He** was a bank clerk in the Yukon before he published "Songs of a Sourdough" in 1907.
> THEATRE: A new play based on **this Sir Arthur Conan Doyle canine classic** opened on the London stage in 2007.

The questions are parsed, named entities are extracted (*Sir Arthur Conan Doyle* identified as a PERSON, Yukon as a GEOPOLITICAL ENTITY, "Songs of a Sourdough" as a COMPOSITION), coreference is run (*he* is linked with *clerk*) and relations like the following are extracted:

> authorof(focus,"Songs of a sourdough")
> publish (e1, he, "Songs of a sourdough")

in (e2, e1, 1907)
temporallink(publish(...), 1907)

focus

Next DeepQA extracts the question **focus**, shown in bold in both examples. The focus is the part of the question that co-refers with the answer, used for example to align with a supporting passage. The focus is extracted by hand-written rules—made possible by the relatively stylized syntax of Jeopardy! questions—such as a rule extracting any noun phrase with determiner "this" as in the Conan Doyle example, and rules extracting pronouns like *she*, *he*, *hers*, *him*, as in the poet example.

lexical answer type

The **lexical answer type** (shown in blue above) is a word or words which tell us something about the semantic type of the answer. Because of the wide variety of questions in Jeopardy!, Jeopardy! uses a far larger set of answer types than the sets for standard factoid algorithms like the one shown in Fig. 28.4. Even a large set of named entity tags is insufficient to define a set of answer types. The DeepQA team investigated a set of 20,000 questions and found that a named entity tagger with over 100 named entity types covered less than half the types in these questions. Thus DeepQA extracts a wide variety of words to be answer types; roughly 5,000 lexical answer types occurred in the 20,000 questions they investigated, often with multiple answer types in each question.

These lexical answer types are again extracted by rules: the default rule is to choose the syntactic headword of the focus. Other rules improve this default choice. For example additional lexical answer types can be words in the question that are coreferent with or have a particular syntactic relation with the focus, such as headwords of appositives or predicative nominatives of the focus. In some cases even the Jeopardy! category can act as a lexical answer type, if it refers to a type of entity that is compatible with the other lexical answer types. Thus in the first case above, *he*, *poet*, and *clerk* are all lexical answer types. In addition to using the rules directly as a classifier, they can instead be used as features in a logisitic regression classifier that can return a probability as well as a lexical answer type.

Note that answer types function quite differently in DeepQA than the purely IR-based factoid question answerers. In the algorithm described in Section 28.1, we determine the answer type, and then use a strict filtering algorithm only considering text strings that have exactly that type. In DeepQA, by contrast, we extract lots of answers, unconstrained by answer type, and a set of answer types, and then in the later 'candidate answer scoring' phase, we simply score how well each answer fits the answer types as one of many sources of evidence.

Finally the question is classified by type (definition question, multiple-choice, puzzle, fill-in-the-blank). This is generally done by writing pattern-matching regular expressions over words or parse trees.

In the second **candidate answer generation** stage, we combine the processed question with external documents and other knowledge sources to suggest many candidate answers. These candidate answers can either be extracted from text documents or from structured knowledge bases.

For structured resources like DBpedia, IMDB, or the triples produced by Open Information Extraction, we can just query these stores with the relation and the known entity, just as we saw in Section 28.2. Thus if we have extracted the relation `authorof(focus,"Songs of a sourdough")`, we can query a triple store with `authorof(?x,"Songs of a sourdough")` to return the correct author.

The method for extracting answers from text depends on the type of text documents. To extract answers from normal text documents we can do passage search

just as we did in Section 28.1. As we did in that section, we need to generate a query from the question; for DeepQA this is generally done by eliminating stop words, and then upweighting any terms which occur in any relation with the focus. For example from this query:

> MOVIE-"ING": Robert Redford and Paul Newman starred in this depression-era grifter flick. *(Answer: "The Sting")*

the following weighted query might be extracted:

> (2.0 Robert Redford) (2.0 Paul Newman) star depression era grifter (1.5 flick)

The query can now be passed to a standard IR system. Some systems are already set up to allow retrieval of short passages, and the system can just return the ten 1-2 sentence passages that are needed for the next stage. Alternatively the query can be passed to a standard document retrieval engine, and then from each returned document passages are selected that are longer, toward the front, and have more named entities.

DeepQA also makes use of the convenient fact that the vast majority of Jeopardy! answers are the title of a Wikipedia document. To find these titles, we can do a second text retrieval pass specifically on Wikipedia documents. Then instead of extracting passages from the retrieved Wikipedia document, we directly return the titles of the highly ranked retrieved documents as the possible answers.

Once we have a set of passages, we need to extract candidate answers. As we just said, if the document is a Wikipedia page, we can just take the title, but for other texts, like news documents, we need other approaches. Two common approaches are to extract all **anchor texts** in the document (anchor text is the text between <a> and <\a> used to point to a URL in an HTML page), or to extract all noun phrases in the passage that are Wikipedia document titles.

anchor texts

The third **candidate answer scoring** stage uses many sources of evidence to score the candidates. One of the most important is the lexical answer type. DeepQA includes a system that takes a candidate answer and a lexical answer type and returns a score indicating whether the candidate answer can be interpreted as a subclass or instance of the answer type. Consider the candidate "difficulty swallowing" and the lexical answer type "manifestation". DeepQA first matches each of these words with possible entities in ontologies like DBpedia and WordNet. Thus the candidate "difficulty swallowing" is matched with the DBpedia entity "Dysphagia", and then that instance is mapped to the WordNet type "Symptom". The answer type "manifestation" is mapped to the WordNet type "Condition". The system looks for a link of hyponymy, instance-of or synonymy between these two types; in this case a hyponymy relation is found between "Symptom" and "Condition".

Other scorers are based on using time and space relations extracted from DBpedia or other structured databases. For example, we can extract temporal properties of the entity (when was a person born, when died) and then compare to time expressions in the question. If a time expression in the question occurs chronologically before a person was born, that would be evidence against this person being the answer to the question.

Finally, we can use text retrieval to help retrieve evidence supporting a candidate answer. We can retrieve passages with terms matching the question, then replace the focus in the question with the candidate answer and measure the overlapping words or ordering of the passage with the modified question.

The output of this stage is a set of candidate answers, each with a vector of scoring features.

In the final **answer merging and scoring** step, we first merge candidate answers that are equivalent. Thus if we had extracted two candidate answers *J.F.K.* and *John F. Kennedy*, this stage would merge the two into a single candidate. For proper nouns, automatically generated name dictionaries can help in this task. One useful kind of resource is the large synonym dictionaries that are created by listing all anchor text strings that point to the same Wikipedia page; such dictionaries give large numbers of synonyms for each Wikipedia title — e.g., *JFK, John F. Kennedy, John Fitzgerald Kennedy, Senator John F. Kennedy, President Kennedy, Jack Kennedy*, etc. (Spitkovsky and Chang, 2012). For common nouns, we can use morphological parsing to merge candidates which are morphological variants.

We then merge the evidence for each variant, combining the scoring feature vectors for the merged candidates into a single vector.

Now we have a set of candidates, each with a feature vector. A regularized logistic regression classifier is used to take each feature vector and assign a single confidence value to this candidate answer. The classifier is trained on thousands of candidate answers, each labeled for whether it is correct or incorrect, together with their feature vectors, and learning to predict a probability of being a correct answer. Since, in training, there are far more incorrect answers than correct answers, we need to use one of the standard techniques for dealing with very imbalanced data. DeepQA uses *instance weighting*, assigning an instance weight of .5 for each incorrect answer example in training. The candidate answers are then sorted by this confidence value, resulting in a single best answer.

The merging and ranking is actually run iteratively; first the candidates are ranked by the classifier, giving a rough first value for each candidate answer, then that value is used to decide which of the variants of a name to select as the merged answer, then the merged answers are re-ranked,.

In summary, we've seen in the four stages of DeepQA that it draws on the intuitions of both the IR-based and knowledge-based paradigms. Indeed, Watson's architectural innovation is its reliance on proposing a very large number of candidate answers from both text-based and knowledge-based sources and then developing a wide variety of evidence features for scoring these candidates —again both text-based and knowledge-based. Of course the Watson system has many more components for dealing with rare and complex questions, and for strategic decisions in playing Jeopardy!; see the papers mentioned at the end of the chapter for many more details.

28.4 Evaluation of Factoid Answers

mean reciprocal rank
MRR

A common evaluation metric for factoid question answering, introduced in the TREC Q/A track in 1999, is **mean reciprocal rank**, or **MRR**. MRR assumes a test set of questions that have been human-labeled with correct answers. MRR also assumes that systems are returning a short **ranked** list of answers or passages containing answers. Each question is then scored according to the reciprocal of the **rank** of the first correct answer. For example if the system returned five answers but the first three are wrong and hence the highest-ranked correct answer is ranked fourth, the reciprocal rank score for that question would be $\frac{1}{4}$. Questions with return sets that do not contain any correct answers are assigned a zero. The score of a system is then the average of the score for each question in the set. More formally, for an evaluation of a system returning a set of ranked answers for a test set consisting of

N questions, the MRR is defined as

$$\text{MRR} = \frac{1}{N} \sum_{i=1 \text{ s.t. } rank_i \neq 0}^{N} \frac{1}{rank_i} \tag{28.9}$$

A number of test sets are available for question answering. Early systems used the TREC QA dataset; questions and hand-written answers for TREC competitions from 1999 to 2004 are publicly available. **FREE917** (Cai and Yates, 2013) has 917 questions manually created by annotators, each paired with a meaning representation; example questions include:

FREE917

How many people survived the sinking of the Titanic?
What is the average temperature in Sydney in August?
When did Mount Fuji last erupt?

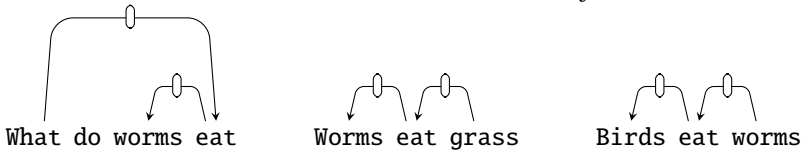
WEBQUESTIONS

WEBQUESTIONS (Berant et al., 2013) contains 5,810 questions asked by web users, each beginning with a wh-word and containing exactly one entity. Questions are paired with hand-written answers drawn from the Freebase page of the question's entity, and were extracted from Google Suggest by breadth-first search (start with a seed question, remove some words, use Google Suggest to suggest likely alternative question candidates, remove some words, etc.). Some examples:

What character did Natalie Portman play in Star Wars?
What airport is closest to Palm Springs?
Which countries share land border with Vietnam?
What present day countries use English as their national language?

Bibliographical and Historical Notes

Question answering was one of the earliest NLP tasks, and early versions of the text-based and knowledge-based paradigms were developed by the very early 1960s. The text-based algorithms generally relied on simple parsing of the question and of the sentences in the document, and then looking for matches. This approach was used very early on (Phillips, 1960) but perhaps the most complete early system, and one that strikingly prefigures modern relation-based systems, was the Protosynthex system of Simmons et al. (1964). Given a question, Protosynthex first formed a query from the content words in the question, and then retrieved candidate answer sentences in the document, ranked by their frequency-weighted term overlap with the question. The query and each retrieved sentence were then parsed with dependency parsers, and the sentence whose structure best matches the question structure selected. Thus the question *What do worms eat?* would match *worms eat grass*: both have the subject *worms* as a dependent of *eat*, in the version of dependency grammar used at the time, while *birds eat worms* has *birds* as the subject:



The alternative knowledge-based paradigm was implemented in the BASEBALL system (Green et al., 1961). This system answered questions about baseball games like "Where did the Red Sox play on July 7" by querying a structured database of game information. The database was stored as a kind of attribute-value matrix with values for attributes of each game:

```
Month = July
   Place = Boston
       Day  = 7
       Game Serial No.  = 96
       (Team = Red Sox, Score = 5)
       (Team = Yankees, Score = 3)
```

Each question was constituency-parsed using the algorithm of Zellig Harris's TDAP project at the University of Pennsylvania, essentially a cascade of finite-state transducers (see the historical discussion in Joshi and Hopely 1999 and Karttunen 1999). Then a content analysis phase each word or phrase was associated with a program that computed parts of its meaning. Thus the phrase 'Where' had code to assign the semantics `Place = ?"`, with the result that the question "Where did the Red Sox play on July 7" was assigned the meaning

```
Place = ?
Team = Red Sox
Month = July
Day = 7
```

The question is then matched against the database to return to the answer. Simmons (1965) summarizes other early QA systems.

Another important progenitor of the knowledge-based paradigm for question-answering is work that used predicate calculus as the meaning representation language. The **LUNAR** system (Woods et al. 1972,Woods 1978) was designed to be a natural language interface to a database of chemical facts about lunar geology. It could answer questions like *Do any samples have greater than 13 percent aluminum* by parsing them into a logical form

LUNAR

(TEST (FOR SOME X16 / (SEQ SAMPLES) : T ; (CONTAIN' X16
(NPR* X17 / (QUOTE AL203)) (GREATERTHAN 13PCT))))

The rise of the web brought the information-retrieval paradigm for question answering to the forefront with the TREC QA track beginning in 1999, leading to a wide variety of factoid and non-factoid systems competing in annual evaluations.

The DeepQA component of the Watson system that won the Jeopardy! challenge is described in a series of papers in volume 56 of the IBM Journal of Research and Development; see for example Ferrucci (2012), Lally et al. (2012), Chu-Carroll et al. (2012), Murdock et al. (2012b), Murdock et al. (2012a), Kalyanpur et al. (2012), and Gondek et al. (2012).

Question answering is also an important function of modern personal assistant dialog systems; see Chapter 29 for more.

Exercises

Berant, J., Chou, A., Frostig, R., and Liang, P. (2013). Semantic parsing on freebase from question-answer pairs. In *EMNLP 2013*.

Berant, J. and Liang, P. (2014). Semantic parsing via paraphrasing. In *ACL 2014*.

Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., and Hellmann, S. (2009). DBpedia—A crystallization point for the Web of Data. *Web Semantics: science, services and agents on the world wide web*, *7*(3), 154–165.

Bollacker, K., Evans, C., Paritosh, P., Sturge, T., and Taylor, J. (2008). Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD 2008*, pp. 1247–1250.

Brill, E., Dumais, S. T., and Banko, M. (2002). An analysis of the AskMSR question-answering system. In *EMNLP 2002*, pp. 257–264.

Cai, Q. and Yates, A. (2013). Large-scale semantic parsing via schema matching and lexicon extension.. In *ACL 2013*, pp. 423–433.

Chang, A. X. and Manning, C. D. (2012). SUTime: A library for recognizing and normalizing time expressions.. In *LREC-12*, pp. 3735–3740.

Chu-Carroll, J., Fan, J., Boguraev, B. K., Carmel, D., Sheinwald, D., and Welty, C. (2012). Finding needles in the haystack: Search and candidate generation. *IBM Journal of Research and Development*, *56*(3/4), 6:1–6:12.

Fader, A., Soderland, S., and Etzioni, O. (2011). Identifying relations for open information extraction. In *EMNLP-11*, pp. 1535–1545.

Fader, A., Zettlemoyer, L., and Etzioni, O. (2013). Paraphrase-driven learning for open question answering. In *ACL 2013*, Sofia, Bulgaria, pp. 1608–1618.

Ferrucci, D. A. (2012). Introduction to "this is watson". *IBM Journal of Research and Development*, *56*(3/4), 1:1–1:15.

Gondek, D., Lally, A., Kalyanpur, A., Murdock, J. W., Duboué, P. A., Zhang, L., Pan, Y., Qiu, Z., and Welty, C. (2012). A framework for merging and ranking of answers in deepqa. *IBM Journal of Research and Development*, *56*(3/4), 14:1–14:12.

Green, B. F., Wolf, A. K., Chomsky, C., and Laughery, K. (1961). Baseball: An automatic question answerer. In *Proceedings of the Western Joint Computer Conference 19*, pp. 219–224. Reprinted in Grosz et al. (1986).

Harabagiu, S., Pasca, M., and Maiorano, S. (2000). Experiments with open-domain textual question answering. In *COLING-00*, Saarbrücken, Germany.

Hovy, E. H., Hermjakob, U., and Ravichandran, D. (2002). A question/answer typology with surface text patterns. In *HLT-01*.

Joshi, A. K. and Hopely, P. (1999). A parser from antiquity. In Kornai, A. (Ed.), *Extended Finite State Models of Language*, pp. 6–15. Cambridge University Press.

Kalyanpur, A., Boguraev, B. K., Patwardhan, S., Murdock, J. W., Lally, A., Welty, C., Prager, J. M., Coppola, B., Fokoue-Nkoutche, A., Zhang, L., Pan, Y., and Qiu, Z. M. (2012). Structured data and inference in deepqa. *IBM Journal of Research and Development*, *56*(3/4), 10:1–10:14.

Karttunen, L. (1999). Comments on Joshi. In Kornai, A. (Ed.), *Extended Finite State Models of Language*, pp. 16–18. Cambridge University Press.

Lally, A., Prager, J. M., McCord, M. C., Boguraev, B. K., Patwardhan, S., Fan, J., Fodor, P., and Chu-Carroll, J. (2012). Question analysis: How Watson reads a clue. *IBM Journal of Research and Development*, *56*(3/4), 2:1–2:14.

Li, X. and Roth, D. (2002). Learning question classifiers. In *COLING-02*, pp. 556–562.

Li, X. and Roth, D. (2005). Learning question classifiers: The role of semantic information. *Journal of Natural Language Engineering*, *11*(4).

Lin, J. (2007). An exploration of the principles underlying redundancy-based factoid question answering. *ACM Transactions on Information Systems*, *25*(2).

Monz, C. (2004). Minimal span weighting retrieval for question answering. In *SIGIR Workshop on Information Retrieval for Question Answering*, pp. 23–30.

Murdock, J. W., Fan, J., Lally, A., Shima, H., and Boguraev, B. K. (2012a). Textual evidence gathering and analysis. *IBM Journal of Research and Development*, *56*(3/4), 8:1–8:14.

Murdock, J. W., Kalyanpur, A., Welty, C., Fan, J., Ferrucci, D. A., Gondek, D. C., Zhang, L., and Kanayama, H. (2012b). Typing candidate answers using type coercion. *IBM Journal of Research and Development*, *56*(3/4), 7:1–7:13.

Pasca, M. (2003). *Open-Domain Question Answering from Large Text Collections*. CSLI.

Phillips, A. V. (1960). A question-answering routine. Tech. rep. 16, MIT AI Lab.

Simmons, R. F. (1965). Answering English questions by computer: A survey. *Communications of the ACM*, *8*(1), 53–70.

Simmons, R. F., Klein, S., and McConlogue, K. (1964). Indexing and dependency logic for answering english questions. *American Documentation*, *15*(3), 196–204.

Spitkovsky, V. I. and Chang, A. X. (2012). A cross-lingual dictionary for English Wikipedia concepts. In *LREC-12*, Istanbul, Turkey.

Woods, W. A. (1978). Semantics and quantification in natural language question answering. In Yovits, M. (Ed.), *Advances in Computers*, pp. 2–64. Academic.

Woods, W. A., Kaplan, R. M., and Nash-Webber, B. L. (1972). The lunar sciences natural language information system: Final report. Tech. rep. 2378, BBN.

Zelle, J. M. and Mooney, R. J. (1996). Learning to parse database queries using inductive logic programming. In *AAAI-96*, pp. 1050–1055.

Zettlemoyer, L. and Collins, M. (2005). Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Uncertainty in Artificial Intelligence, UAI'05*, pp. 658–666.