Introduction to "This is Watson"

D. A. Ferrucci

In 2007, IBM Research took on the grand challenge of building a computer system that could compete with champions at the game of Jeopardy!™. In 2011, the open-domain question-answering (QA) system, dubbed Watson, beat the two highest ranked players in a nationally televised two-game Jeopardy! match. This paper provides a brief history of the events and ideas that positioned our team to take on the Jeopardy! challenge, build Watson, IBM Watson™, and ultimately triumph. It describes both the nature of the QA challenge represented by Jeopardy! and our overarching technical approach. The main body of this paper provides a narrative of the DeepQA processing pipeline to introduce the articles in this special issue and put them in context of the overall system. Finally, this paper summarizes our main results, describing how the system, as a holistic combination of many diverse algorithmic techniques, performed at champion levels, and it briefly discusses the team's future research plans.

Introduction

The open-domain question-answering (QA) problem is one of the most challenging in the realm of computer science and artificial intelligence (AI). QA has had a long history [1] and has seen considerable advancement over the past decade [2, 3].

Jeopardy!** is a well-known television quiz show that has been on air in the United States for more than 25 years. It pits three human contestants against one another in a competition that requires rapidly understanding and answering rich natural-language questions, which are called *clues*, over a very broad domain of topics, with stiff penalties for wrong answers [4]. On January 14, 2011, at IBM Research in Yorktown Heights, New York, IBM Watson*, a computer, beat the two best Jeopardy! champions in a real-time two-game competition. The historic match was conducted and taped by Jeopardy Productions, Inc. and was nationally televised over three nights on February 14–16, 2011.

The fact that a computer beat the best human contestants at Jeopardy! represents a major landmark in open-domain QA, but in many ways, this is just the beginning. Research in open-domain QA requires advances in many areas of computer science and AI, including information retrieval (IR), natural-language processing (NLP), knowledge representation and reasoning (KR&R), machine learning, and human-computer interfaces (HCIs). Techniques that bring all these technologies together to process language and knowledge have a long way to go before computers can interact and reason over natural-language content at human levels.

What we have accomplished with Watson is the development of a software architecture and a methodology that builds on, integrates, and advances decades of innovation in these fields. It demonstrates a capability few thought possible, and one that compelled us forward when many might have given up. Watson's public performance has opened the door to commercial applications and a future where scientists and businesspeople alike have a deeper appreciation for the potential impact of technologies that promise to tap into the wealth of knowledge buried in text and other unstructured data sources.

Many high-level descriptions of Watson and its performance on Jeopardy! have appeared in the popular press. An overview paper about DeepQA, which is the underlying architecture and technology powering Watson, was published in *AI Magazine* before the match was played [4], but few papers have been published detailing the internal algorithms used in Watson and how they relate to prior work. This special issue of the *IBM Journal of Research and Development* collects a set of articles describing many of Watson's internal algorithms and how they were combined to deliver a winning performance at the Jeopardy! task.

Digital Object Identifier: 10.1147/JRD.2012.2184356

0018-8646/12/$5.00 © 2012 IBM

The purpose of this paper is to introduce this special issue and describe the different articles contained herein. Considering the general interest and significance of the accomplishment represented by Watson, this paper first provides a brief history of the events and ideas that positioned our team to take on the Jeopardy! challenge, build Watson, and ultimately triumph. I describe the nature of the technical challenges represented by Jeopardy! and discuss our overarching technical approach. In the main body of this paper, I use a narrative of the DeepQA processing pipeline to introduce the articles in this journal and put them in context of the overall system. Finally, I summarize the main results, how the system, as a holistic combination of many diverse algorithmic techniques, was tuned to perform at champion levels and the team's future research plans.

History

Unstructured information

Much of human communication, whether it is in natural-language text, speech, or images, is *unstructured*. The semantics necessary to interpret unstructured information to solve problems is often implicit and must be derived by using background information and inference. With structured information, such as traditional database tables, the data is well-defined, and the semantics is explicit. Queries are prepared to answer predetermined questions on the basis of necessary and sufficient knowledge of the meaning of the table headings (e.g., Name, Address, Item, Price, and Date). However, what does an arbitrary string of text or an image really mean? How can a computer program act on the content of a "note" or a "comment" without explicit semantics describing its intended meaning or usage?

With the enormous proliferation of electronic content on the web and within our enterprises, unstructured information (e.g., text, images, and speech) is growing far faster than structured information. Whether it is general reference material, textbooks, journals, technical manuals, biographies, or blogs, this content contains high-value knowledge essential for informed decision making. The promise of leveraging the knowledge latent in these large volumes of unstructured text lies in deeper natural-language analysis that can more directly infer answers to our questions.

NLP techniques, which are also referred to as text analytics, infer the meaning of terms and phrases by analyzing their syntax, context, and usage patterns. Human language, however, is so complex, variable (there are many different ways to express the same meaning), and polysemous (the same word or phrase may mean many things in different contexts) that this presents an enormous technical challenge. Decades of research have led to many specialized techniques, each operating on language at different levels and on different isolated aspects of the language understanding task. These techniques include, for example, shallow parsing, deep parsing, information extraction, word-sense disambiguation [5, 6], latent semantic analysis [7], textual entailment [8], and coreference resolution [9]. None of these techniques is perfect or complete in their ability to decipher the intended meaning. Unlike programming languages, human languages are not formal mathematical constructs. Given the highly contextual and implicit nature of language, humans themselves often disagree about the intended meaning of any given expression.

Our interpretations of text can be heavily influenced by personal background knowledge about the topic or about the writer or about when the content was expressed. Consider the expression "That play was bad!" What sort of thing does "play" refer to? A play on Broadway? A football play? Does "bad" mean "good" in this sentence? Clearly, more context is needed to interpret the intended meaning accurately. Language processing techniques, such as natural-language parsing, coreference resolution, or word-sense disambiguation, need to independently advance. However, perhaps most importantly, we need to understand how to integrate them as contributing components within more comprehensive language understanding systems rather than advance them only in response to isolated evaluations. How good does a parser have to be if different techniques over large data resources in a larger system help get the answer?

In more comprehensive systems such as open-domain QA and dialogue systems, we can learn how many different algorithms interact and complement each other to perform complex tasks. Such system-level experiments facilitate a deeper understanding of which algorithmic techniques require more investment in order to more rapidly advance the field as a whole. End-to-end system research is difficult, however. Just getting collections of independently developed analytics to talk to each other and then scale efficiently enough to run large experiments is a challenge.

From 2001 through 2006, we built the Unstructured Information Management Architecture (UIMA) to facilitate this kind of basic interoperability. UIMA is a software architecture and framework that provides a common platform for integrating diverse collections of text, speech, and image analytics independently of algorithmic approach, programming language, or underlying domain model [10]. UIMA is focused on the general notion of integrating a scalable set of cooperating software programs, called annotators, which assign semantics to some region of text (or image or speech). In 2006, IBM contributed UIMA to Apache [11], and it is currently in regular use around the world by industry and academia.

UIMA provides the essential infrastructure needed to engage large-scale language understanding research. Open-domain QA for Jeopardy! became the driving challenge problem. Building Watson was the system-level experiment that brought together hundreds of different

cooperating algorithms. Reminiscent of Minsky's "Society of Mind" [12], each of these algorithms alone performs relatively simple language processing tasks. None completely understands the question or can single-handedly justify an answer, and none alone behaves like a smart Jeopardy! player, but combined as part of the DeepQA architecture, they enable Watson to perform as a highly competitive Jeopardy! champion.

Open-domain QA and the Jeopardy! challenge

In May 1997, IBM's Deep Blue* computer beat Gary Kasparov, the reigning chess grand champion, in a six-game chess match. The extremely broad appeal of this project and the global focus it brought to the science performed at IBM Research beckoned for an encore. Could a Jeopardy! playing machine be the next Deep Blue for IBM? The problem, of course, was that open-domain QA presented a vastly different challenge in computer science than did chess. For humans, it was clear that to be a chess master, you have to be really smart. To win at Jeopardy!, you have to know a lot, be well read, understand language and its nuances, and think and react quickly. Nonetheless, if asked which task was more difficult, I suspect that most people would say chess. However, whereas chess is a well-defined task with explicit, complete, and precise mathematical semantics, natural language is ill-defined and ambiguous, and its meaning is implicit.

From 2004, the year Ken Jennings lit up the airwaves with his record 74-game Jeopardy! winning streak, through the end of 2006, the head of IBM Research challenged researchers to build a system that could win at Jeopardy!. Most believed it was impossible—pure folly destined to embarrass IBM and to destroy the credibility of all the researchers who attempted it. For me, and ultimately my team, taking on Jeopardy! was absolutely irresistible.

We had an accomplished history in the open-domain QA task. Our prior QA work took shape in the form of a QA system we called PIQUANT [13, 14]. PIQUANT development started in 1999, predating our work in UIMA, and was funded by government research grants and tested against NIST (National Institute of Standards and Technology) evaluation data in the Text REtrieval Conference (TREC) QA track between 1999 and 2005 [15]. PIQUANT was based on state-of-the-art techniques that used a single ontology and combined parsing, information extraction, and search to generate answers to factoid-style questions. It consistently scored in the top tier of systems in TREC evaluations. Although our work on PIQUANT advanced our NLP technology in key areas and resulted in many published papers, its performance was far from what would be required to compete in Jeopardy!.

PIQUANT, like most state-of-the-art QA systems at the time, presumed a static predetermined set of answer types—classes of concepts that the question was asking for (e.g., people, places, dates, and numbers). The breadth of domain (types of things asked about) and complexity of language in Jeopardy! exceeded that of TREC evaluations. The demand for accurate probabilities in an answer's correctness and the demand for speed also far exceeded that of TREC evaluations. Moreover, with PIQUANT, there was no easy way to extend, rapidly measure, and advance with a diversity of techniques contributed by different researchers.

I believed, though, that with the right resources fully dedicated to the challenge, even failure would have taught us more about the field than years of incremental progress. At the end of 2006, we received the support to conduct a four-month feasibility study. The conclusion was that the Jeopardy! challenge would drive key innovations and push the science and engineering of open-domain QA to new heights.

A whole new architecture, technical approach, and culture had to be put in place. Whereas UIMA facilitated the technical integration and scale-out of a broad range of analytic components, the challenge for building Watson became the development of the algorithms themselves and the architecture and methodology necessary for rapidly combining, measuring, and advancing them to succeed at Jeopardy!. In April 2007, we committed to the challenge of building an open-domain QA capability, which is good enough in terms of precision, confidence estimation, and speed, to compete against grand champions at Jeopardy! within three to five years.

What it takes to win at Jeopardy!

To compete on Jeopardy!, Watson would have to act like a real-time Jeopardy! contestant. It would have to play real-time games with new clues never before seen by Watson or its developers. Like its human competitors, it would have to be completely self-contained—no web search, no connection to the Internet, and no interaction with anyone else for help in understanding or answering the questions. Before the game, it would have to "study"—*analyze* and store in its memory every bit of information it might consider during the game.

In an average of just three seconds, Watson would have to parse a clue, understand what it was asking for, relate its meaning to what it had "read," determine the best answer, and compute whether it was confident enough to buzz in. It would then have to buzz in fast enough to attempt an answer; speak the answer; and finally, based on the result, select the next clue or give up control to another player.

A keyword search engine can deliver millions of potentially relevant documents. Where's the correct answer? Is it even there? Even if the answer is somewhere in a top hit, what word or phrase or combination thereof represents the exact answer, and how do you determine the likelihood it

is correct? If Watson gives a wrong answer, just like any other Jeopardy! contestant, it loses the dollar value of the clue. Watson would need to produce its best answer from among hundreds of possibilities. Moreover, it would need to consistently determine an accurate probability that its best answer was indeed correct. Programming a computer to achieve the necessary precision and confidence over natural-language questions requires that the computer perform deeper *analysis* of the clue and of all the content it has read that might justify an answer.

To estimate how good humans are at this task, we gathered and analyzed data from approximately 2,000 former Jeopardy! games. For each available game, we counted the total number of questions attempted by the game's winner. These are the questions a player is confident enough to buzz in for and fast enough to get a chance to answer. Then, we counted the percentage of attempted questions the winner correctly answered. We found that, on average, winning players attempt between 40% and 50% of the questions in a game and get between 85% and 95% correct. Because there is competition for the buzz, a player will not always get a chance to answer all the questions he wants to answer. Therefore, depending on the competition, to answer 40% to 50% of the questions, a player must be confident enough to buzz in for more questions.

On the basis of this analysis, we set a performance target that would enable Watson to buzz in for at least 70% of the questions in a game and of those, get at least 85% correct. We called this level of performance *85% Precision at 70% answered*, or simply *85% Precision@70*. This aggressive performance target would put Watson in a competitive position against champion players. Of course, to win at a real-time game, Watson would also have to compute its answers and confidences fast enough to be competitive at the buzz and would have to make good betting decisions on Daily Doubles and Final Jeopardy! questions.

Inside DeepQA: The architecture underlying Watson

In 2007, we created a baseline system using PIQUANT and measured its performance on a Jeopardy! test set. It delivered about 16% Precision@70. This and other early baselines we developed in 2007 failed to approach the precision and confidence estimation required to even qualify for the game, much less compete with grand champions. We needed to dramatically revamp our approach. Over the next year, two important technical artifacts emerged from our work that provided the foundation for our success with Watson: first, an extensible software architecture for building QA systems, which we named DeepQA, and second, a methodology based on the ideas in [16] for the rapid advancement and integration of many core algorithmic techniques, which we named AdaptWatson [17].

DeepQA—The DeepQA architecture is illustrated in **Figure 1**, and a high-level description appears in [4]. The architecture defines various stages of analysis in a processing pipeline. Each stage admits multiple implementations that can produce alternative results. At each stage, alternatives are independently pursued as part of a massively parallel computation. DeepQA never assumes that any component perfectly understands the question and can just look up the right answer in a database. Rather, many candidate answers are proposed by searching many different resources, on the basis of different interpretations of the question and category. A commitment to any one answer is deferred while more and more evidence is gathered and analyzed for each answer and each alternative path through the system.

DeepQA applies hundreds of algorithms that analyze evidence along different dimensions, such as type classification, time, geography, popularity, passage support, source reliability, and semantic relatedness. This analysis produces hundreds of features or scores, each indicating the degree to which a bit of evidence supports an answer according to one of these dimensions. All the features for a candidate answer must be combined into a single score representing the probability of the answer being correct. DeepQA trains statistical machine learning algorithms on prior sets of questions and answers to learn how best to weight each of the hundreds of features relative to one another. These weights are used at run time to balance all of the features when combining the final scores for candidate answers to new questions. The final result of the process is a ranked list of candidate answers, each with a final confidence score representing the likelihood the answer is correct based on the analysis of all its supporting evidence. If the top answer's confidence is above a threshold, Watson wants to answer. If not, it will not take the chance.

Below, I use the DeepQA processing pipeline as a backdrop for introducing the papers in this issue that describe many of the algorithms the team has developed to implement each processing stage.

AdaptWatson—By the end of 2007, we had an initial implementation of the DeepQA framework. It admitted plug-ins from the team of researchers and performed the full end-to-end QA task on Jeopardy! clues. Although its performance was poor, the team was in a position to incrementally advance core algorithms; measure results; and, based on those results, come up with new ideas and iterate. Over the next year, a methodology named AdaptWatson and a set of tools emerged for rapidly advancing the research, development, integration, and evaluation of more than 100 core algorithmic components. These components were designed to understand questions, search for candidate answers, collect evidence, score evidence and answers, produce confidences, and merge and rank results.

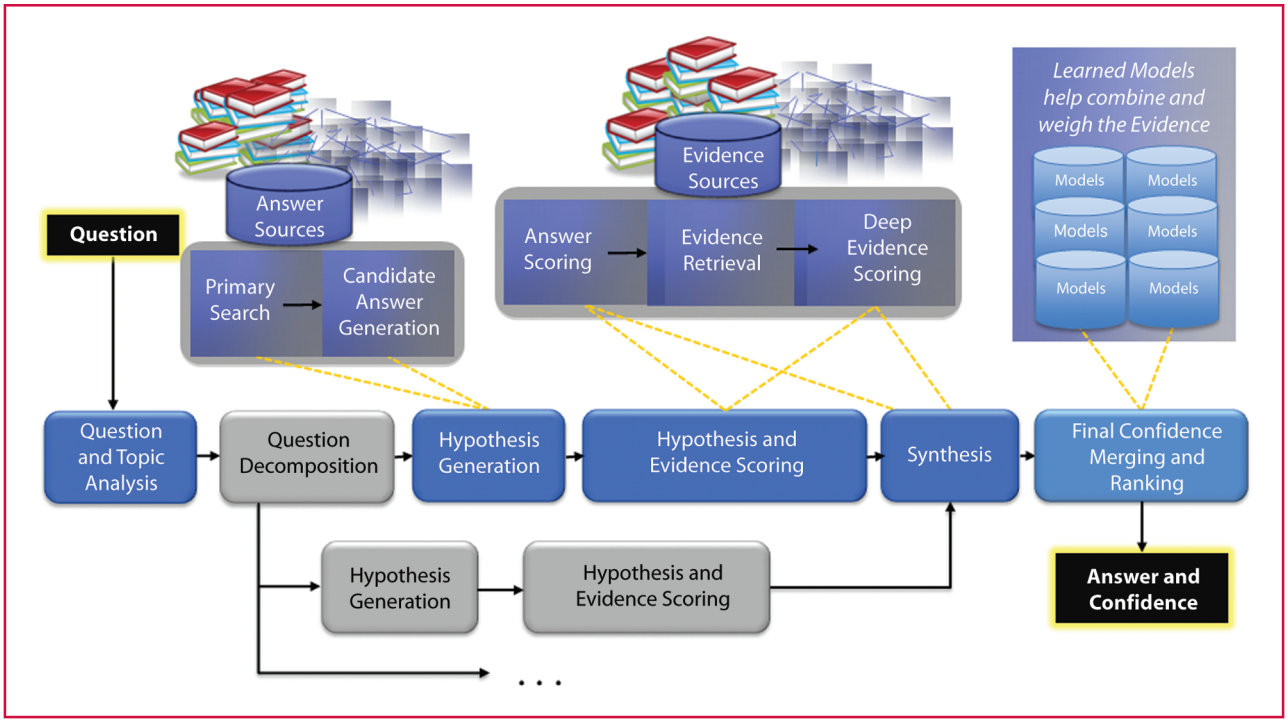

Figure 1

DeepQA architecture.

At the same time, the research team grew to about 25 full-time researchers and engineers, including several student members from key university partnerships. The team performed and documented more than 8,000 independent experiments by the time Watson went live. Each experiment generated 10 to 20 GB of trace data. Tools were developed to efficiently explore this data and discover failures and their likely causes. On the basis of analysis of this data, the team generated new algorithmic ideas and quantitatively estimated their potential impact on end-to-end performance. This data was used to prioritize, develop, and test new algorithms. Successful algorithmic advances were included in biweekly full-system builds. These were regularly run to produce updated baseline performance. This iterative process was implemented by the core team of researchers working in a single room and supported by more than 200 eight-core servers.

With the DeepQA architecture and the AdaptWatson methodology in place, the team drove the performance of Watson from early baselines delivering roughly 20% Precision@70 to greater than 85% Precision@70—good enough to compete with champions. Many of the papers in this issue describe the result of advancing core algorithms based on using DeepQA as a foundational architecture and the AdaptWatson methodology as a team-oriented process for rapidly creating and advancing a wide diversity of algorithm techniques to meet target performance.

Understanding questions

The breadth of the Jeopardy! domain is exemplified by the richness of language used, the variety of questions asked, and the huge range of types and topics covered. It is a challenge just to analyze the questions well enough to determine *what* they might be asking for or *how* the focus of the clue relates to other key elements in the clue. The more precisely Watson understands the clue, the better chance it has at finding and justifying answers.

We refer to the word or phrase that indicates the *class* of thing the clue is asking for as the *lexical answer type*, or LAT. The clue in the first example below is asking for a president, which is a useful LAT. However, the LAT in the subsequent clue—"they"—does not carry much semantic information at all. The third clue below claims to be looking for a "star," but, in fact, the answer is a unique synthesis of Tom Cruise and cruise control—no star at all.

RECENT HISTORY: President under whom the U.S. gave full recognition to Communist China. (Answer: "Jimmy Carter")

POP MUSIC: Their grandfather had a number 1 record in 1935; their father, Number 1's in 1958 & 1961; and they hit number 1 in 1990.
(Answer: "Gunnar & Matthew Nelson")

BEFORE & AFTER: The "Jerry Maguire" star who automatically maintains your vehicle's speed.
(Answer: "Tom Cruise control")

In the prior examples, the category was not required to identify the LAT. In the following clue, however, analysis of the category "FOLLOW THE LEADER TO THE CAPITAL" is essential to figure out what the question is asking for:

FOLLOW THE LEADER TO THE CAPITAL: Pervez Musharraf. (Answer: "Islamabad")

The first technical paper in this special issue, "Question Analysis: How Watson Reads a Clue," by Lally et al. [18], starts at the beginning of the DeepQA processing pipeline. It describes Watson's question analysis algorithms—how it analyzes a clue and attempts to determine what it is asking about and the kind of thing it is asking for. The second paper, "Deep Parsing in Watson," by McCord et al. [19], presents ESG, our English Slot Grammar parser and the predicate-argument structure (PAS) generator. ESG produces a grammatical parse of a sentence and identifies parts of speech and syntactic roles such as subject, predicate, and object, as well as modification relations between sentence segments. The PAS generator is used to produce a more abstract representation of a parse suitable for other analytics further down the DeepQA processing pipeline. Natural-language parsing techniques are critical for analyzing questions and all the content Watson uses to discover answers and their supporting evidence.

Answer and evidence sources

Watson had to be completely self-contained. Like the human contestants, it could not access the Internet or any resources outside the confines of its physical "body." Part of the challenge was to determine what resources Watson should consider as a source for answers and evidence. We started with the obvious—encyclopedias and reference books—and then developed a semiautomatic process for growing Watson's content. There are several tradeoffs to deal with. First, adding content could hurt system performance if it contributed more noise than value. Second, adding more content was the biggest driver for hardware requirements. Although we were dedicated to building a system capable of competing at Jeopardy!, ultimately, there were limits. We could have discovered, for example, that the hardware requirements were just too great to meet practical concerns relating to cost, space, and efficiency. We could not indiscriminately add servers and therefore, could not indiscriminately grow the content.

The next two papers, "Textual Resource Acquisition and Engineering," by Chu-Carroll et al. [20], and "Automatic Knowledge Extraction from Documents," by Fan et al. [21], describe our approach to acquiring, evaluating, integrating, and expanding Watson's content. The first paper discusses how raw textual content was selected, evaluated, and automatically expanded. The second discusses our approach to analyzing the raw content for building a derived resource, called PRISMATIC. PRISMATIC uses parsing, information extraction, and statistics to induce axiomatic knowledge from text for later use in candidate generation and in answer scoring. For example, automatically induced and included in PRISMATIC are commonsense axioms, such as "books are found on shelves," "people visit museums," "people visit websites," and "candidates win elections." These axioms can help generate and score plausible answers.

Discovering candidate answers

In spite of the high accuracy obtained with our natural-language parser, producing a perfect logical representation of a clue and then finding the identical representation in some structured database that contains an answer was extremely rare. This failing was due to the wide variety in expression of language and the breadth of topics, type categories, and properties in Jeopardy!, relative to the very limited expression and narrow coverage of existing structured resources such as DBpedia [22]. Our best attempts at leveraging structured resources for producing answers resulted in confident answers less than 2% of the time [4].

DeepQA does not assume that it ever completely understands the question with respect to any preexisting model, set of type ontologies, or database schema. Rather, the results of parsing and question analysis result in multiple interpretations of the question and, ultimately, in a variety of different queries. These queries are run against different structured and unstructured sources using a variety of search mechanisms with complementary strengths and weaknesses. Rather than attempt to directly answer the question, the idea at this point in the pipeline is to first generate a broad set of *candidate answers*. Each candidate answer combined with the question represents an independent hypothesis. Each becomes the root of an independent process that attempts to discover and evaluate supporting evidence in its candidate answer. In "Finding Needles in the Haystack: Search and Candidate Generation," Chu-Carroll et al. [23] provide a description of the search and candidate generation components used in Watson to find possible answers. A metric referred to as *candidate binary recall* is computed as the percentage of questions for which the correct answer is generated as a candidate. This metric reflects the goal of maximizing candidate recall at the hypothesis generation phase of the DeepQA pipeline.

Right type

If candidate generation finds a correct answer for only 85% of the questions, then Watson can, at most, get 85% of the questions correct. It can only perform this well if, after considering all of the candidates, it ranks the right answer in the first position and with enough confidence to take the risk and buzz in. Accurately computing a probability that a candidate is correct is a critical and very challenging feature of the system. Buzzing in with a wrong answer is costly. A contestant will lose the dollar value of that clue and end up helping his competitors earn the money on the rebound. The entire remainder of Watson's processing pipeline and a significant portion of its computational resources are spent on finding the best answer and computing an accurate probability that it might be correct.

To do that, Watson analyzes many different classes of evidence for each answer. An important class of evidence considered is whether the answer is of the right answer type. It was clear from early experiments that the approach we took in PIQUANT, where we anticipated all answer types and built specialized algorithms for finding instances of anticipated types (such as people, organizations, and animals), would not be sufficient. Jeopardy! refers to far too many types (on the order of thousands) for us to have considered developing *a priori* information extraction components for each one. Moreover, they are used in ways where their meaning is highly contextual and difficult to anticipate. For example, in the first clue below, the *Running_Foot_St_EvenLAT is "quality." How many things might be correctly classified as a "quality?" Similarly, the next two questions are both asking for a "direction."

> ART HISTORY: Unique quality of "First Communion of Anemic Young Girls in the Snow," shown at the 1883 Arts Incoherents Exhibit.
> (Answer: "all white")
>
> DECORATING: If you're standing, it's the direction you should look to check out the wainscoting.
> (Answer: "down")
>
> SEWING: The direction of threads in a fabric is called this, like the pattern of fibers in wood.
> (Answer: "grain")

Jeopardy! uses very opened-ended types. We employ dynamic and flexible techniques for classifying candidate answers, heavily relying on the context in the question. For this, we developed a technique we called *type coercion* [24]. Type coercion radically differs from earlier systems (such as PIQUANT) that statically classify a possible answer on the basis of a preexisting set of types. Instead, it takes a lexical answer type such as "direction" and poses a type hypothesis for each candidate, such as *is_a* ("down," "direction") or *is_a* ("grain," "direction"). It then consults a wide variety of structured and unstructured resources using a diverse set of algorithmic techniques to gather evidence for and against the type hypothesis.

No component in the system was pretrained in classifying "directions." Rather, Watson uses an extensible ensemble of techniques based on a wide variety of textual and ontological resources, including PRISMATIC [21], YAGO [25], and WordNet** [26]. Algorithms designed to use these resources independently quantify a degree of support for believing that the type hypothesis is true. In the next paper, in the issue, "Typing Candidate Answers using Type Coercion," Murdock et al. [27] discuss the type coercion techniques developed for Watson that provide the dynamic classification necessary for dealing with the huge variety of contextual types used in Jeopardy!.

Collecting and scoring evidence

After DeepQA generates candidate answers and confidence scores that indicate the degree to which each answer is considered to be an instance of the answer type, DeepQA attempts to collect and *score* additional evidence. Algorithms whose function is to score evidence are called *evidence scorers*. These algorithms are designed to produce a *confidence score*—a number that indicates the degree to which a piece of evidence supports or refutes the correctness of a candidate answer. Multiple evidence scorers can work in parallel for each candidate answer and over different forms of evidence. One type of evidence is *passage evidence*—paragraphs of text found in volumes of textual resources that might support the correctness of a candidate answer. In "Textual Evidence Gathering and Analysis," Murdock et al. [28] discuss the methods used to gather and score passage evidence using grammar-based and other syntactic analysis techniques.

Grammar-based techniques address syntactic and shallow semantic features of language. They look for how the words and structure of language may predict similarities in meaning. Relation extraction techniques look deeper into the intended meaning, attempting to find semantic relationships (e.g., starred in, visited, painted, invented, and naturalized in) between concepts, although they may have been expressed with different words or with different grammatical structures. In "Relation Extraction and Scoring in DeepQA," Wang et al. [29] present two approaches to broad-domain relation extraction and scoring: one based on manual pattern specification (rule based) and the other relying on statistical methods for pattern elicitation, which uses a novel transfer learning technique, *relation topics*.

The rule-based approach is more precise and is used by several components, but it requires manual effort to develop patterns for a small targeted set of relations

(approximately 30). Statistical approaches, on the other hand, automatically learn how to extract semantic relations from training data, which may be collected semiautomatically and were trained to detect occurrences of a large number of relations (approximately 7,000). Below, we show three examples of clues and the different relations that were extracted.

MOTHERS & SONS: Though only separated by one year in real life, she played mother to son Colin Farrell in "Alexander." (Answer: "Angelina Jolie")

- Relation: starring (she, "Alexander")

THE DEVIL: "The Screwtape Letters" from a senior devil to an under devil are by this man better known for children's books.
(Answer: "C(live) S(taples) Lewis")

- Relation: author (man, "The Screwtape Letters")

THE LONE REPRESENTATIVE: Michael Castle from this state with 3 counties: New Castle, Kent, and Sussex. (Answer: "Delaware")

- Relation: residence ("Michael Castle", state)

Because of the breadth of topics and variability in language, and despite the success of relationship extraction, DeepQA can rarely reduce the question to a single concise semantic relation, as in *directed_film* (*person*, "*Green Mansions*"), and then accurately map it to some existing structured database schema, for example, using DBpedia [22], Freebase [30], or the Internet Movie Database (IMDB) [31], where it can correctly look up the answer (less than 2% of the time). Structured databases tend to be too narrow in their coverage and too brittle in their expression. Their schemas capture one view of the data and do not attempt to reflect the enormous range of linguistic expressions that may map to their data.

Although structured resources tend to be too narrow and brittle to directly and confidently answer questions, several components in DeepQA do use structured data, such as databases, knowledge bases, and ontologies, to generate potential candidate answers or find and score additional evidence on the basis of different features of the question context. In "Structured Data and Inference in DeepQA," Kalyanpur et al. [32] discuss several areas in which evidence from structured sources has the most impact, include typing answers, geospatial and temporal constraints, and frames that formally encode *a priori* knowledge of commonly appearing entity types, such as countries and U.S. presidents.

The first example below illustrates the use of structured sources to determine that the "Black Hole of Calcutta" is an Asian location—an example of a geospatial constraint. In the next example, by identifying the topic area and the frame elements involved, the frame subsystem assumes that the country being asked for is a country whose national language is Khmer. The third example highlights that the frame subsystem enumerates all entities of the given type (U.S. presidents and countries) and looks up the relevant characteristics.

THE HOLE TRUTH: Asian location where a notoriously horrible event took place on the night of June 20, 1756. (Answer: "Black Hole of Calcutta")

LANGUAGE: The lead singer of the band Dengue Fever is from this country & often sings in Khmer. (Answer: "Cambodia")

PRESIDENTS: The only 2 consecutive U.S. presidents with the same first name.
(Answer: "James Madison and James Monroe")

Puzzles, Final Jeopardy!, and other Special Questions

In a significant number of cases, Jeopardy! clues can be nothing like ordinary fact-seeking questions. Consider Special Questions such as the following:

ARE YOU A FOOD "E"?: Escoffier says to leave them in their shells & soak them in a mixture of water, vinegar, salt & flour. (Answer: "Escargots")

ASTRONOMICAL RHYME TIME: Any song about Earth's natural satellite. (Answer: "moon tune")

BEFORE & AFTER: The "Jerry Maguire" star who automatically maintains your vehicle's speed.
(Answer: "Tom Cruise control")

SHAKESPEAREAN ANAGRAMS: She's "one girl" King Lear should have been leery of.
(Answer: "Goneril")

Special Questions are handled by a collection of algorithms designed to first detect their occurrence and then to favor the algorithmic paths in the DeepQA architecture specialized for finding and synthesizing answers. Although the vast majority of the work on DeepQA was focused on general IR, NLP, KR&R, and machine learning techniques, a small part of the overall effort was focused on Jeopardy!-specific techniques. Special Questions were part of that effort. In the next paper, "Special Questions and Techniques," Prager et al. [33] focus on question types that require special processing, including puzzles and puns.

Missing links

Consider the following clues known in Jeopardy! as "COMMON BONDS":

> COMMON BONDS: feet, eyebrows, and McDonald's. (Answer: "arches")

> COMMON BONDS: trout, loose change in your pocket, and compliments.
> (Answer: "things that you fish for")

Realizing and resolving implicit relationships and using them to interpret language and to answer questions is generally useful and appears in different forms in Jeopardy! clues. Although these examples are fun and may seem of unique interest to Jeopardy!, the ability to find what different concepts have in common can help in many areas, including relating symptoms to diseases or generating hypotheses that a chemical might be effective at treating a disease. For example, a hypothesis of using fish oil as a treatment for Raynaud's disease was made after text analytics discovered that both had a relationship to blood viscosity [34].

In "COMMON BONDS" clues, the task of finding the missing link is directly suggested by the well-known Jeopardy! category. However, this is not always the case. Final Jeopardy! questions, in particular, were uniquely difficult, partly because they made implicit references to unnamed entities or missing links. The missing link had to be correctly resolved in order to evidence the right answer. Consider the following Final Jeopardy! clue:

> EXPLORERS: On hearing of the discovery of George Mallory's body, he told reporters he still thinks he was first. (Answer: "Sir Edmund Hillary")

To answer this question accurately, Watson had to first make the connection to Mount Everest and realize that, although not the answer, it is essential to confidently getting the correct answer—in this case, Edmund Hillary, the first person to reach the top of Mount Everest. Implicit relationships and other types of tacit context that help in interpreting language are certainly not unique to Jeopardy! but are commonplace in ordinary language. In the next paper in this journal issue, "Identifying Implicit Relationships," Chu-Carroll et al. [35] discuss the algorithmic techniques used to solve "COMMON BONDS" questions, as well as other questions, such as the Final Jeopardy! question above, which require the discovery of missing links and implicit relationships.

Breaking the question down

Another important technique generally useful for QA is breaking a question down into logical subparts, so that the subparts may be independently explored and the results combined to produce the answer. Consider that to produce an answer for the clue below, the system might first determine that the "1831 work" (which includes the two chapters mentioned) is the "Hunchback of Notre Dame" and then solve for the author of this book. Nested questions such as this require decompositions to be processed in sequence, with the answer to an inner subquestion plugged into the outer part.

> WORLD AUTHORS: Chapters in an 1831 work by this author include "Maitre Jacques Coppenole" & "A Tear for a Drop of Water."
> (Answer: "Victor Hugo")

Parallel decomposable questions contain subquestions that can be evaluated independently of each other and the evidence combined. For example, in the clue below, the system can independently find characters introduced in 1894 and then words that come from Hindi for "bear." The system can build confidence in a single answer that independently satisfies both subquestions. In "Fact-Based Question Decomposition in DeepQA," Kalyanpur et al. [36] discuss techniques for accurately identifying decomposable questions and then decomposing them into useful and relevant subparts and the role these algorithms play in improving Watson's performance.

> FICTIONAL ANIMALS: The name of this character, introduced in 1894, comes from the Hindi for "bear."
> (Answer: "Baloo")

Merging evidence and combining confidence

As an example, Watson may consider 100 candidate answers for some question. For each of these, it may find 100 pieces of evidence in the form of paragraphs or facts from databases. Each evidence-answer pair may be scored by 100 independent scorers. Each scoring algorithm produces a confidence. For any one candidate, there may be on the order of 10,000 confidence scores—on the order of one million in total for a single question.

The final stage in the DeepQA pipeline is dedicated to ranking all candidate answers according to their evidence scores and judging the likelihood that each candidate answer is correct. Its job is to figure out the best way to combine the confidence of many different scorers across different pieces of evidence to produce a single probability for each candidate answer. This is done using a statistical machine learning framework. The framework in DeepQA is phase-based, providing capabilities for manipulating the data and applying machine learning in successive applications to deal with issues such as normalization, training with sparse training data, and merging related or equivalent candidate answers.

The learning approach facilitates an agile development environment for DeepQA. Because machine learning is used to weigh and combine scores, evidence scoring algorithms can be easily introduced, revised, and reconfigured without the need for an error-prone manual effort to determine how to combine the various evidence scores. In "A Framework for Merging and Ranking of Answers in DeepQA," Gondek et al. [37] describe the machine learning framework used in DeepQA, explain the challenges, and evaluate the gains over a baseline machine learning approach.

Massive parallelism, scale-out, and speed

By the end of 2008, the DeepQA architecture and our methodology for rapidly advancing the performance seemed to be working. We were far from champion-level performance, but we were steadily progressing. Planning for success required that we begin to consider *latency*—the time it took to answer a question. To compete with the best players at Jeopardy!, Watson would have to produce its best answer and confidence estimation in an average of about three seconds. At the end of 2008, DeepQA took about two hours of CPU time to answer a Jeopardy! question, running on a single 2.6-GHz processor with 16 GB of memory. This latency would have made for a very boring Jeopardy! game.

Leveraging DeepQA's implementation on UIMA, we employed a new five-person subteam dedicated to scaling DeepQA by embedding it in UIMA-AS [38]. UIMA-AS allows any UIMA application to be deployed as a collection of asynchronous processes that use standard messaging infrastructure to communicate. Each can be independently scaled up, as needed, by deploying multiple instances of the process on a larger number of machines.

From the beginning, we designed DeepQA to implement an embarrassingly parallel process. First, each search query could be independently executed. Subsequently, each candidate answer could start a new independent thread of processing, where each piece of evidence for each candidate answer, could be independently scored by more than 100 different algorithms. This parallel computation was perfectly suited for the type of scale-out supported by UIMA-AS. In "Making Watson Fast," Epstein et al. [39] describe how DeepQA was scaled out using UIMA-AS using 2,880 processors to drive the QA latency down from two hours to three seconds.

Watson: An artificial Jeopardy! contestant

DeepQA refers to the architecture and implementation of a system that takes a question and a category as input and provides a ranked list of answers. Each answer is associated with probabilities that the answer is correct, based on analyzed evidence. DeepQA does not play Jeopardy!. To play Jeopardy!, DeepQA was embedded in a program that transformed it into Watson, a real-time Jeopardy! contestant. This involved two principal components: one to handle the game strategy and the other to interface with the game itself, i.e., to speak a clue selection, get the clue, press the buzzer, and keep track of the game.

The strategy component does three things as follows.

1. Decides *which* clues to select when Watson has control of the board. This decision is based on prior confidences, likely positions of Daily Doubles, and Watson's ability to adjust its prior confidences on the basis of revealed answers to former clues in a Jeopardy! category.
2. Determines *what* to bet on Daily Doubles and on Final Jeopardy!. This decision is based on prior confidences and the state of the game, including how much of the game has been played and all of the players' scores, and player modeling.
3. Determines a *confidence threshold* below which Watson will not buzz in (even if it could) and above which Watson will attempt to buzz in. This determination is based on a risk assessment. For example, if Watson has a commanding lead, the strategy component may determine that answering is not worth the risk of getting the question wrong, in which case, it will raise the threshold to 98% confidence.

Although the effort to create Watson's strategy component was dwarfed by the effort needed to create the DeepQA engine, very advanced techniques were developed to produce effective clue-selection, betting, and threshold-setting strategies. Once the DeepQA engine was smart and fast enough to compete with champions, the difference in winning and losing could come down to pure strategy. In the paper, "Simulation, Learning, and Optimization Techniques in Watson's Game Strategies," Tesauro et al. [40] describe the methods and algorithms used to develop and train Watson's strategy component.

Finally, to play Jeopardy!, Watson had to "view" the board, keep track of the game status, consider other players scores, "read" a clue, speak its clue selections, speak its answers, and press down on a handheld buzzer. A system was built to interface the DeepQA engine and the strategy component with the official Jeopardy! game control system. We were focused on the natural-language QA task and opted not to address visual or auditory clues, so Watson, in effect, could not see or hear. Jeopardy! had made allowances for blind contestants in the past. Our goal was to find a set of equivalent interfaces to the Jeopardy! control system to keep a level playing field. This does not mean that Watson was handicapped in any way to make it more human-like. It is a computer. What it did mean was that it would not have a practical advantage in interfacing with the game system. Hence, for example, where human players had to push a button to buzz in, Watson could not send

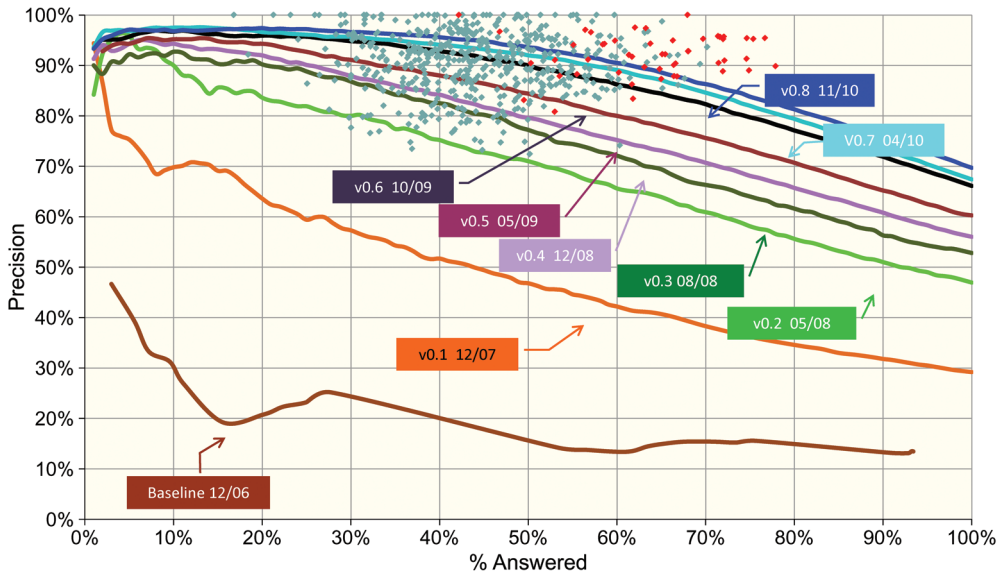

Figure 2

Incremental progress in answering precision on the Jeopardy! challenge: June 2007 to November 2011.

an electronic signal directly to the Jeopardy! controller; it too would have to mechanically push the button. In the final paper, "In the Game: The Interface between Watson and Jeopardy!," Lewis [41] presents the details of the system that interfaces Watson with the Jeopardy! game control system, making it a fully functional Jeopardy! contestant.

Summary of results

After four years of effort by an average of roughly 25 core researchers and engineers, Watson was ready to play. We relied on different ways to measure our results: 1) precision and confidence, 2) game-winning performance, and 3) component performance.

The first two sets of metrics are end-to-end system metrics. They consider the performance of the whole QA system on the Jeopardy! task. They are best represented by plotting QA precision against percentage answered to produce a confidence curve. **Figure 2** shows the confidence curves for successive versions of the system, starting with the PIQUANT-based baseline system and progressing through the various versions of DeepQA. The *x*-axis gives the percentage of questions answered, and the *y*-axis gives precision (i.e., for those questions answered, the percentage correctly answered). Each DeepQA curve is produced by running DeepQA on 200 games' worth of blind data. These games contain 12,000 questions that were never viewed by the developers and never used to train the system. The data set was used only to test performance on roughly a quarterly basis.

For comparison, Figure 2 also illustrates the performance of champion Jeopardy! players. Each blue dot plots the performance of the winner of an official televised Jeopardy! game, showing the percentage of questions answered by the winner and the winner's precision. The red dots in the plot show the same information for games won by Ken Jennings—the player who had the longest winning streak on Jeopardy! and who was ultimately defeated by Brad Rutter in a Tournament of Champions match. Collectively, we refer to this set of dots as the *winner's cloud*. For Watson to be competitive with champion players, the DeepQA confidence curve must cut through the top half of the winner's cloud. The final confidence curve plotted in Figure 2 (labeled "v0.8 11/10") did just that. The curve illustrates that over 200 games' worth of blind data, Watson delivered an average of 92% precision between 40% and 70% attempted and over 85% Precision@70. This QA performance gave Watson the ability to compete with top Jeopardy! players but certainly did not guarantee a win.

The second measurement, which is also an end-to-end measurement, includes the impact of strategy and speed in formal real-time gameplay with original Jeopardy! equipment, against champion-level competition. Before the game that aired on national television, Watson was pitted against former Jeopardy! Tournament of Champions players. All of these players are arguably in the same league as

Table 1 DeepQA technology performance on public benchmark sets. (ACE: automatic content extraction; RTE: recognizing textual entailment.)

NLP task	*Evaluation set*	*Project start*	*State of art*	*Watson*
Parsing	Wikipedia** accuracy	84.4	81.1 Charniak parser [19]	88.7
Entity disambiguation	Wikipedia disambiguation F_1	72.5	81.9 Hoffart et al. [42]	92.5
Relation detection	ACE 2004 F_1	45.8	72.1 Zhang et al. [43]	73.2
Textual entailment	RTE-6 2010 F_1	34.6	48.0 PKUTM [44]	48.8

Ken Jennings and Brad Rutter and, on any given day, may defeat either one of them. Watson played 55 real-time previously unseen games against these players and won 71% of them. To do this, Watson computed its confidences and its best answer in approximately three seconds, on average, and included a very competitive betting strategy.

The third set of metrics is distributed across the individual component algorithms that populate the DeepQA processing pipeline. Each paper in this special issue presents individual component technologies and describes how they affect end-to-end performance. However, it is important to realize that the quantity and diversity of components used to implement DeepQA make it extraordinarily robust and flexible. The system does not heavily depend on any one of them. We have run many ablation experiments consistently showing that all but the most dramatic of ablations have very little effect.

For example, when we run experiments, in which we ablate a single evidence scorer from the full Watson system, we rarely see a statistically significant impact on a few thousand questions, and we never see an impact of 1% or greater. However, if we ablate *all* of the evidence scorers, this heavily ablated version of Watson answers only 50% correctly. It produces a confidence curve that is insufficient to compete at Jeopardy!. Consequently, to illuminate and measure the effectiveness of individual components, the papers in this journal describe a variety of different approaches.

One important method used in several of the papers relies on a simpler configuration of Watson we have built called the Watson answer-scoring baseline (WASB) system. The WASB system includes all of Watson's question analysis, search, and candidate generation components. It includes only one evidence-scoring component: an answer-typing component that uses a named-entity detector. The use of named-entity detection, to determine whether a candidate answer has the semantic type that the question requires, is a very popular technique in QA (as discussed in detail in [27]), which is why we included it in our baseline. We have evaluated the impact of adding various components to the WASB system [27–29, 32] and found that we are able to examine and compare the individual effectiveness of components for which simple ablations from the full system do not provide statistically meaningful insights given the size of the test sets we use. We see some individual components that provide an impact on accuracy in the range of 2% to 5% when added to the WASB system.

Jeopardy! proved to be an excellent challenge problem. The goal to beat human champions drove the creation and advancement of the DeepQA architecture and the AdaptWatson methodology. Both proved indispensable for conducting large-scale open-domain QA research and, more generally, language understanding research. In addition to winning against human champions at Jeopardy!, the project allowed us to produce leading component-level metrics on core NLP technologies, including parsing, relation detection, named-entity disambiguation, and textual entailment. These results are summarized in **Table 1**. For each task, we report the performance of the technology used in Watson on public benchmark sets, which, in each case, leads over state-of-the-art comparisons found in [19, 42–44]. We also compare against our performance on these same tasks at the beginning of the project, which shows that working toward the Jeopardy! challenge helped drive substantial gains and fundamental improvements in our NLP capabilities that extend beyond Jeopardy!.

Future directions

Watson, as developed for Jeopardy!, attempts to provide a single correct answer and associated confidence. We would like to see applications of the DeepQA technology move toward a broader range of capabilities that engage in dialogues with users to provide decision support over large volumes of unstructured content. The notion of a computer system helping to produce and provide evidence for alternative solutions has been around for decades. Such knowledge-based decision support tools, however, traditionally suffer from the requirement to manually craft and encode formal logical models of the target domain, such as medicine, where these models represent the concepts, their relationships, and the rules that govern their interaction. It can be prohibitively inefficient to do this for broad bodies of knowledge. It is slow and expensive to maintain these formal structures as the raw knowledge grows and as

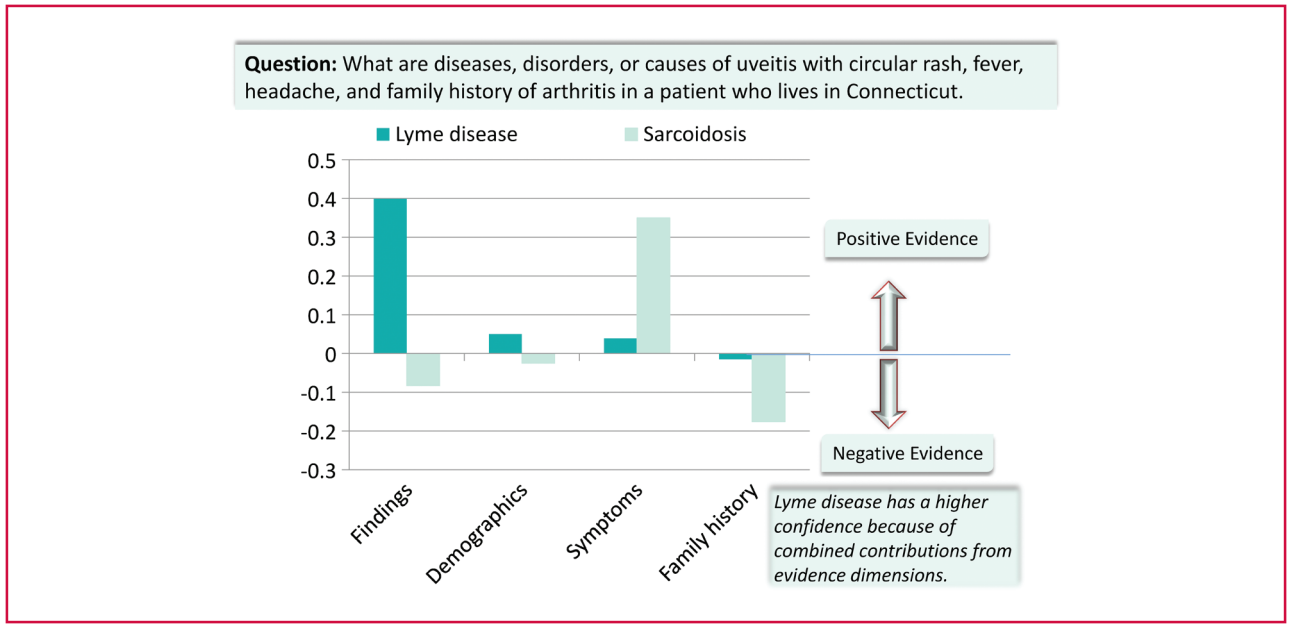

Figure 3

Evidence profiles for differential diagnosis in medicine.

the ways in which different user groups view the domain and formulate queries evolve.

We envision developing Watson 2.0 as an interactive decision support capability that strikes a balance somewhere between a search system and a formal knowledge-based system. The goal is to consider not simply queries but also entire problem scenarios, as well as to produce hypothetical solutions and compelling evidence, not by manually producing a rich and complete model of the domain beforehand but rather by automatically consuming and analyzing natural-language sources of knowledge. We imagine a result that can deliver value in evidence-based decision support over enormous breadths of knowledge at much lower costs.

Healthcare is an exciting area to drive this technical vision. Many problem-solving situations, such as diagnosing a medical problem and then treating it, require viewing the patient's case as a problem scenario. Patients and caregivers are overwhelmed with hoards of ever-changing data represented in mostly unstructured resources, and they are losing confidence that they have efficient enough access to the knowledge they need to rationalize important medical decisions—whether it be finding the right diagnosis or justifying the best treatment options. Efficient personalized analysis of options and evidence sourced in reference material, textbooks, journals, and structured resources can empower patients and caregivers alike, reducing costs and improving patient outcomes. To advance DeepQA and take Watson into healthcare, we plan to move from taking specific questions as input to analyzing entire *problem scenarios*, represented, for example, by a patient's electronic medical record. This will require a richer set of analytics that must discover potential problems in the electronic medical record and generate meaningful questions to help in diagnosis or treatment.

Watson 2.0 will have to produce *evidence profiles* and *supporting evidence* that enable the user to explore the specific evidence justifying a diagnostic hypothesis or treatment alternatives, in support of securing more informed decisions. The notion of an evidence profile is described by Ferrucci et al. [45]. Adapting evidence profiles for the medical domain would allow caregivers to explore the justifications for answers in terms of relevant evidential dimensions. An example, illustrated in **Figure 3**, shows a proposed set of clinical dimensions of evidence for a patient from Connecticut, with a chief complaint consisting of eye pain and inflammation, blurred vision, headache, fever, and circular rash. Each dimension of evidence—*findings*, *demographics*, *symptoms*, and *family history*—aggregates individual pieces of evidence. A healthcare provider can observe the contribution of each dimension of evidence to the overall confidence score, as well as drill-down into a particular dimension to evaluate the efficacy of each contributing piece of evidence.

Jeopardy! represents a constrained interaction model, where the system could not engage in a dialogue to resolve

ambiguities or ask for additional information that might improve its confidence. We imagine Watson 2.0 will support *mixed-initiative dialogue* that extends over time as a problem case is explored. This will allow the system to more efficiently prune its search by gathering specific information from the user. For instance, in the process of answering a question, Watson may identify a gap in its understanding and generate *learning questions* that help it fill in knowledge about language and meaning in a particular domain. Watson, for example, could ask its user the following question: Does the phrase *food would* "*get stuck when eating*" suggest that the patient has "*swallowing difficulty*"? This provides an opportunity for Watson to immediately improve its analysis and learn from its users' responses for future use.

Although Watson's win in Jeopardy! represents a landmark success, there is a way to go before we see systems that fluently engage in dialogue and reason over natural-language content. Where the path to this future may have been uncertain, our work with DeepQA has set a course and given us tremendous confidence that we may get there yet.

Acknowledgments

The author would like to thank the courageous, talented, and uniquely committed team of researchers and engineers that he had the distinct pleasure to work with and lead in the creation of DeepQA and Watson. These are the people who designed, built, and continue to advance Watson and its underlying technologies. Their work and the work of many university collaborators, including the University of Massachusetts at Amherst; University of Texas at Austin; University of Trento; MIT; University of Southern California, Information Sciences Institute; and, most notably, the team under Eric Nyberg at Carnegie Mellon University, made Watson a success. The author would like to particularly thank Bill Murdock, one of the principal researchers who contributed to Watson's development, a coauthor of many papers in this issue of the *IBM Journal of Research and Development*, and the one who helped to compile and edit the papers in this journal.

References

1. R. F. Simmons, "Natural language question-answering systems: 1969," *Commun. ACM*, vol. 13, no. 1, pp. 15–30, Jan. 1970.
2. M. Maybury, *New Directions in Question-Answering*. Menlo Park, CA: AAAI Press, 2004.
3. T. Strzalkowski and S. Harabagiu, *Advances in Open-Domain Question-Answering*. Berlin, Germany: Springer-Verlag, 2006.
4. D. Ferrucci, E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. Kalyanpur, A. Lally, J. W. Murdock, E. Nyberg, J. Prager, N. Schlaefer, and C. Welty, "Building Watson: An Overview of the DeepQA Project," *AI Mag.*, vol. 31, no. 3, pp. 59–79, Fall 2010.
5. D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition,* 2nd ed. Upper Saddle River, NJ: Prentice-Hall, 2008.
6. C. Manning and H. Schütze, *Foundations of Statistical Natural Language Processing*. Cambridge, MA: MIT Press, 1999.
7. S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *J. Amer. Soc. Inf. Sci.*, vol. 41, no. 6, pp. 391–407, Sep. 1990.
8. PASCAL, *Recognizing Textual Entailment Challenge*, first RTE challenge 2006 collection. It is now incorporated as part of Text Analytics Conference (TAC). [Online]. Available: http://pascallin.ecs.soton.ac.uk/Challenges/RTE/; http://www.nist.gov/tac/2011/RTE/
9. N. Ge, J. Hale, and E. Charniak, "A statistical approach to anaphora resolution," in *Proc. 6th Workshop Very Large Corp.*, 1998, pp. 417–434.
10. D. Ferrucci and A. Lally, "Building an example application with the unstructured information management architecture," *IBM Syst. J.*, vol. 43, no. 3, pp. 455–475, Jul. 2004.
11. Apache UIMA. [Online]. Available: http://uima.apache.org/
12. M. Minsky, *The Society of Mind*. New York: Simon and Schuster, Mar.1988.
13. J. M. Prager, J. Chu-Carroll, and K. Czuba, "A multi-strategy, multi-question approach to question answering," in *New Directions in Question-Answering*, M. Maybury, Ed. Menlo Park, CA: AAAI Press, 2004.
14. J. M. Prager, J. Chu-Carroll, E. W. Brown, and K. Czuba, "Question answering by predictive annotation," in *Advances in Open-Domain Question-Answering*, T. Strzalkowski and S. Harabagiu, Eds. Berlin, Germany: Springer-Verlag, 2006.
15. J. Chu-Carroll, K. Czuba, P. Duboue, and J. Prager, "IBM's PIQUANT II in TREC 2005," in *Proc. 14th TREC*, 2006, pp. 1–8.
16. D. Ferrucci, E. Nyberg, J. Allan, K. Barker, E. Brown, J. Chu-Carroll, A. Ciccolo, P. Duboue, J. Fan, D. Gondek, E. Hovy, B. Katz, A. Lally, M. McCord, P. Morarescu, B. Murdock, B. Porter, J. Prager, T. Strzalkowski, C. Welty, and W. Zadrozny, "Towards the open advancement of question answering systems," IBM, Armonk, NY, IBM Res. Rep., RC24789. [Online]. Available: http://domino.watson.ibm.com/library/CyberDig.nsf/papers/D12791EAA13BB952852575A1004A055C/$File/rc24789.pdf
17. D. Ferrucci and E. Brown, "AdaptWatson: A methodology for developing and adapting Watson technology," IBM, Armonk, NY, IBM Res. Rep., RC25244, 2011.
18. A. Lally, J. M. Prager, M. C. McCord, B. K. Boguraev, S. Patwardhan, J. Fan, P. Fodor, and J. Chu-Carroll, "Question analysis: How Watson reads a clue," *IBM J. Res. & Dev.*, vol. 56, no. 3/4, Paper 2, pp. 2:1–2:14, May/Jul. 2012.
19. M. C. McCord, J. W. Murdock, and B. K. Boguraev, "Deep parsing in Watson," *IBM J. Res. & Dev.*, vol. 56, no. 3/4, Paper 3, pp. 3:1–3:15, May/Jul. 2012.
20. J. Chu-Carroll, J. Fan, N. Schlaefer, and W. Zadrozny, "Textual resource acquisition and engineering," *IBM J. Res. & Dev.*, vol. 56, no. 3/4, Paper 4, pp. 4:1–4:11, May/Jul. 2012.
21. J. Fan, A. Kalyanpur, D. C. Gondek, and D. A. Ferrucci, "Automatic knowledge extraction from documents," *IBM J. Res. & Dev.*, vol. 56, no. 3/4, Paper 5, pp. 5:1–5:10, May/Jul. 2012.
22. C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann, "DBpedia–A crystallization point for the web of data," *J. Web Semantics*, vol. 7, no. 3, pp. 154–165, Sep. 2009.
23. J. Chu-Carroll, J. Fan, B. K. Boguraev, D. Carmel, D. Sheinwald, and C. Welty, "Finding needles in the haystack: Search and candidate generation," *IBM J. Res. & Dev.*, vol. 56, no. 3/4, Paper 6, pp. 6:1–6:12, May/Jul. 2012.

24. A. Kalyanpur, J. W. Murdock, J. Fan, and C. Welty, "Leveraging community-built knowledge for type coercion in question answering," in *Proc. 10th ISWC*, vol. Part II, L. Aroyo, C. Welty, H. Alani, J. Taylor, and A. Bernstein, Eds., vol. Part II. Berlin, Germany: Sringer-Verlag, pp. 144–156, 2011.
25. F. M. Suchanek, G. Kasneci, and G. Weikum, "YAGO—A core of semantic knowledge," in *Proc. 16th Int. World Wide Web Conf.*, 2007, pp. 697–706.
26. C. Fellbaum, *WordNet: An Electronic Lexical Database*. Cambridge, MA: MIT Press, 1998.
27. J. W. Murdock, A. Kalyanpur, C. Welty, J. Fan, D. A. Ferrucci, D. C. Gondek, L. Zhang, and H. Kanayama, "Typing candidate answers using type coercion," *IBM J. Res. & Dev.*, vol. 56, no. 3/4, Paper 7, pp. 7:1–7:13, May/Jul. 2012.
28. J. W. Murdock, J. Fan, A. Lally, H. Shima, and B. K. Boguraev, "Textual evidence gathering and analysis," *IBM J. Res. & Dev.*, vol. 56, no. 3/4, Paper 8, pp. 8:1–8:14, May/Jul. 2012.
29. C. Wang, A. Kalyanpur, J. Fan, B. K. Boguraev, and D. C. Gondek, "Relation extraction and scoring in DeepQA," *IBM J. Res. & Dev.*, vol. 56, no. 3/4, Paper 9, pp. 9:1–9:12, May/Jul. 2012.
30. K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor, "Freebase: A collaboratively created graph database for structuring human knowledge," in *Proc. SIGMOD ACM SIGMOD Int. Conf. Manage. Data*, 2008, pp. 1247–1250.
31. *IMDB: The Internet Movie Database*. [Online]. Available: http://www.imdb.com/
32. A. Kalyanpur, B. K. Boguraev, S. Patwardhan, J. W. Murdock, A. Lally, C. Welty, J. M. Prager, B. Coppola, A. Fokoue-Nkoutche, L. Zhang, Y. Pan, and Z. M. Qiu, "Structured data and inference in DeepQA," *IBM J. Res. & Dev.*, vol. 56, no. 3/4, Paper 10, pp. 10:1–10:14, May/Jul. 2012.
33. J. M. Prager, E. W. Brown, and J. Chu-Carroll, "Special questions and techniques," *IBM J. Res. & Dev.*, vol. 56, no. 3/4, Paper 11, pp. 11:1–11:13, May/Jul. 2012.
34. D. R. Swanson, "Fish oil, Raynaud's syndrome, and undiscovered public knowledge," *Perspectives Biol. Med.*, vol. 30, no. 1, pp. 7–18, Autumn 1986.
35. J. Chu-Carroll, E. W. Brown, A. Lally, and J. W. Murdock, "Identifying implicit relationships," *IBM J. Res. & Dev.*, vol. 56, no. 3/4, Paper 12, pp. 12:1–12:10, May/Jul. 2012.
36. A. Kalyanpur, S. Patwardhan, B. K. Boguraev, A. Lally, and J. Chu-Carroll, "Fact-based question decomposition in DeepQA," *IBM J. Res. & Dev.*, vol. 56, no. 3/4, Paper 13, pp. 13:1–13:11, May/Jul. 2012.
37. D. C. Gondek, A. Lally, A. Kalyanpur, J. W. Murdock, P. Duboue, L. Zhang, Y. Pan, Z. M. Qiu, and C. Welty, "A framework for merging and ranking of answers in DeepQA," *IBM J. Res. & Dev.*, vol. 56, no. 3/4, Paper 14, pp. 14:1–14:12, May/Jul. 2012.
38. *UIMA Asynchronous Scaleout*, The Apache Software Foundation, Apache UIMA Development Community, ver. 2.3.1, 2011. [Online]. Available: http://uima.apache.org/d/uima-as-2.3.1/uima_async_scaleout.html
39. E. A. Epstein, M. I. Schor, B. S. Iyer, A. Lally, E. W. Brown, and J. Cwiklik, "Making Watson fast," *IBM J. Res. & Dev.*, vol. 56, no. 3/4, Paper 15, pp. 15:1–15:12, May/Jul. 2012.
40. G. Tesauro, D. C. Gondek, J. Lenchner, J. Fan, and J. M. Prager, "Simulation, learning, and optimization techniques in Watson's game strategies," *IBM J. Res. & Dev.*, vol. 56, no. 3/4, Paper 16, pp. 16:1–16:11, May/Jul. 2012.
41. B. L. Lewis, "In the game: the interface between Watson and Jeopardy!," *IBM J. Res. & Dev.*, vol. 56, no. 3/4, Paper 17, pp. 17:1–17:6, May/Jul. 2012.
42. J. Hoffart, M. A. Yosef, I. Bordino, H. Fürstenau, M. Pinkal, M. Spaniol, B. Taneva, S. Thater, and G. Weikum, "Robust disambiguation of named entities in text," in *Proc. EMNLP*, 2011, pp. 782–792.
43. M. Zhang, J. Zhang, J. Su, and G. Zhou, "A composite kernel to extract relations between entities with both flat and structured features," in *Proc. 21st Int. Conf. Comput. Linguistics, 44th Annu. Meeting ACL-44*, Stroudsburg, PA, 2006, pp. 825–832.
44. H. Jia, X. Huang, T. Ma, X. Wan, and J. Xiao, "PKUTM participation TAC 2010 RTE and summarization track," in *Proc. TAC*, 2010, pp. 1–12. [Online]. Available: http://www.nist.gov/tac/publications/2010/participant.papers/PKUTM.proceedings.pdf
45. D. Ferrucci, A. Levas, S. Bagchi, D. Gondek, and E. T. Mueller, "Watson: Beyond Jeopardy!," *J. Artif. Intell.,* 2012.

Received December 14, 2011; accepted for publication December 16, 2011

David A. Ferrucci *IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA (ferrucci@us.ibm.com)*. Dr. Ferrucci is an IBM Fellow and the Principal Investigator for the DeepQA Watson/Jeopardy! project. He has been at the T. J. Watson Research Center since 1995, where he leads the Semantic Analysis and Integration Department. Dr. Ferrucci focuses on technologies for automatically discovering meaning in natural-language content and using it to enable better human decision making. He graduated from Manhattan College, with a B.S. degree in biology and from Rensselaer Polytechnic Institute, in 1994 with a Ph.D. degree in computer science specializing in knowledge representation and reasoning. He has published papers in the areas of AI, knowledge representation and reasoning, natural-language processing, and automatic question answering.