ALGORITHMS FOR INFORMATION RETRIEVAL

Entity Linking Models

Entity Linking Models

- Before we turn to the second major paradigm for question answering, knowledge-based question answering, we introduce the important core technology of entity linking, since it is required for any knowledge-based QA algorithm.

- Entity linking is the task of associating a mention in text with the representation of some real-world entity in an ontology.

- Entity linking is done in (roughly) two stages: **mention detection** and **mention disambiguation**.

Entity Linking Models - Linking based on Anchor Dictionaries and Web Graph

- As a simple baseline we introduce the TAGME linker for Wikipedia, which itself draws on earlier algorithms.
- Wikification algorithms define the set of entities as the set of Wikipedia pages, so we'll refer to each Wikipedia page as a unique entity e.
- TAGME first creates a catalog of all entities (i.e. all Wikipedia pages, removing some disambiguation and other meta-pages) and indexes them in a standard IR engine like Lucene.
- For each page e, the algorithm computes an in-link count in(e): the total number of in-links from other Wikipedia pages that point to e.
- These counts can be derived from Wikipedia dumps. Finally, the algorithm requires an anchor dictionary.

Entity Linking Models - Linking based on Anchor Dictionaries and Web Graph

- An anchor dictionary lists for each Wikipedia page, its anchor texts: the hyperlinked spans of text on other pages that point to it.
- For example, the web page for Stanford University, http://www.stanford.edu, might be pointed to from another page using anchor texts like Stanford or Stanford University:
  Stanford University
- We compute a Wikipedia anchor dictionary by including, for each Wikipedia page e, e's title as well as all the anchor texts from all Wikipedia pages that point to e.
- For each anchor string a we'll also compute its total frequency freq(a) in Wikipedia (including non-anchor uses), the number of times a occurs as a link (which we'll call link(a)), and its link probability linkprob(a) = link(a)/freq(a).

Entity Linking Models - Mention Detection

Given a question (or other text we are trying to link), TAGME detects mentions by querying the anchor dictionary for each token sequence up to 6 words. This large set of sequences is pruned with some simple heuristics (for example pruning substrings if they have small linkprobs). The question

When was Ada Lovelace born?

might give rise to the anchor Ada Lovelace and possibly Ada, but substrings spans like Lovelace might be pruned as having too low a linkprob, and but spans like born have such a low linkprob that they would not be in the anchor dictionary at all.

Entity Linking Models - Mention Disambiguation

- If a mention span is unambiguous (points to only one entity/Wikipedia page), we are done with entity linking!
- However, many spans are ambiguous, matching anchors for multiple Wikipedia entities/pages.
- The TAGME algorithm uses two factors for disambiguating ambiguous spans, which have been referred to as prior probability and relatedness/coherence. The first factor is **p(e|a)**, the probability with which the span refers to a particular entity. For each page **e ∈ E(a)**, the probability p(e|a) that anchor a points to e, is the ratio of the number of links into e with anchor text a to the total number of occurrences of a as an anchor:

$$\text{prior}(a \to e) = p(e|a) = \frac{\text{count}(a \to e)}{\text{link}(a)}$$

Entity Linking Models - Mention Disambiguation

Let's see how that factor works in linking entities in the following question:

What Chinese Dynasty came before the Yuan?

- The most common association for the span Yuan in the anchor dictionary is the name of the Chinese currency, i.e., the probability p(Yuan currency| yuan) is very high.
- Rarer Wikipedia associations for Yuan include the common Chinese last name, a language spoken in Thailand, and the correct entity in this case, the name of the Chinese dynasty.
- So if we chose based only on p(e|a) , we would make the wrong disambiguation and miss the correct link, Yuan dynasty.

Entity Linking Models - Mention Disambiguation

- To help in just this sort of case, TAGME uses a second factor, the relatedness of this entity to other entities in the input question. In our example, the fact that the question also contains the span Chinese Dynasty, which has a high probability link to the page Dynasties in Chinese history, ought to help match Yuan dynasty.

- Given a question q, for each candidate anchors span a detected in q, we assign a relatedness score to each possible entity $e \in E(a)$ of a. The relatedness score of the link $a \rightarrow e$ is the weighted average relatedness between e and all other entities in q. Two entities are considered related to the extent their Wikipedia pages share many in-links

Entity Linking Models - Mention Disambiguation

The relatedness between two entities A and B is computed as

$$\text{rel}(A,B) = \frac{\log(\max(|\text{in}(A)|,|\text{in}(B)|)) - \log(|\text{in}(A) \cap \text{in}(B)|)}{\log(|W|) - \log(\min(|\text{in}(A)|,|\text{in}(B)|))}$$

The vote given by anchor b to the candidate annotation a $\longrightarrow$ X is the average, over all the possible entities of b, of their relatedness to X, weighted by their prior probability:

$$\text{vote}(b,X) = \frac{1}{|\mathcal{E}(b)|} \sum_{Y \in \mathcal{E}(b)} \text{rel}(X,Y)p(Y|b)$$

Entity Linking Models - Mention Disambiguation

The total relatedness score for a → X is the sum of the votes of all the other anchors detected in q:

$$\text{relatedness}(a \rightarrow X) = \sum_{b \in \mathcal{X}_q \setminus a} \text{vote}(b, X)$$

To score a → X, we combine relatedness and prior by choosing the entity X that has the highest relatedness(a → X), finding other entities within a small of this value, and from this set, choosing the entity with the highest prior P(X|a). The result of this step is a single entity assigned to each span in q.

Entity Linking Models - Mention Disambiguation

The TAGME algorithm has one further step of pruning spurious anchor/entity pairs, assigning a score averaging link probability with the coherence.

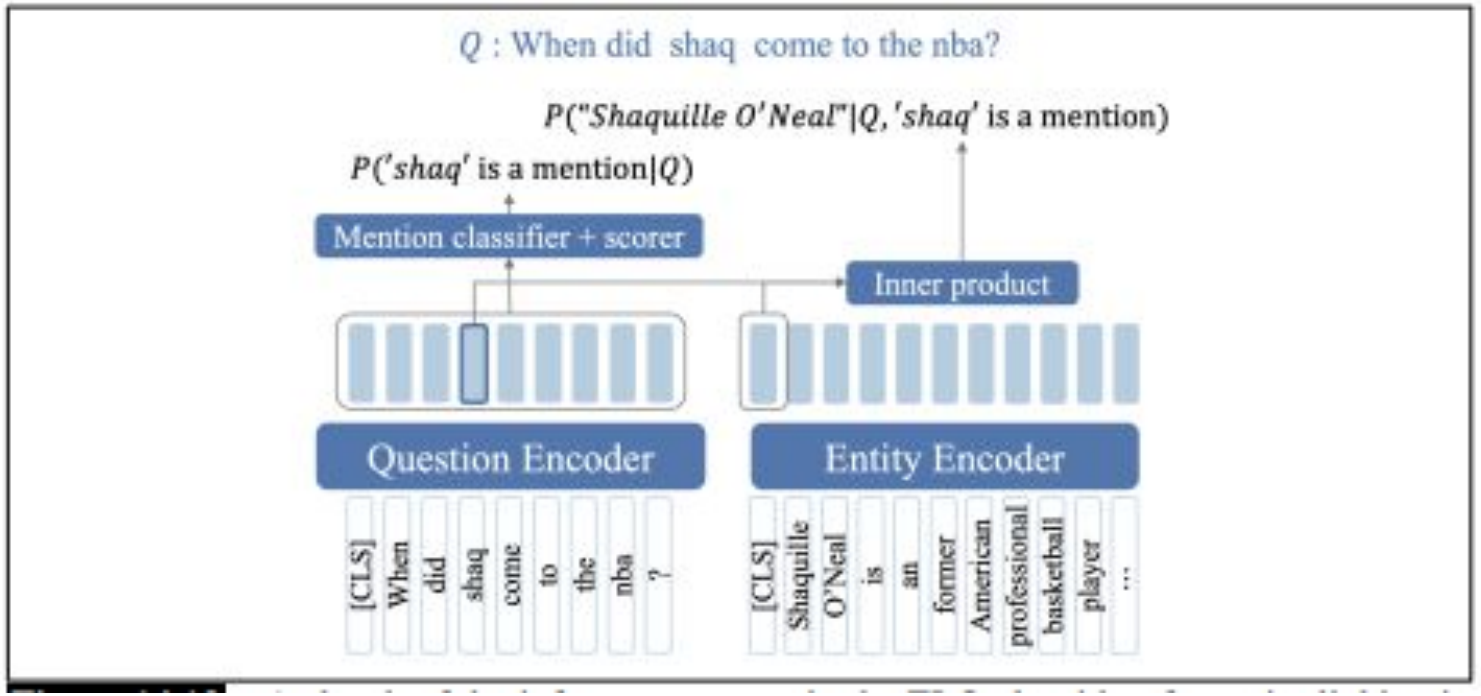
$$\text{coherence}(a \to X) = \frac{1}{|S| - 1} \sum_{B \in \mathcal{S} \setminus X} \text{rel}(B, X)$$

$$\text{score}(a \to X) = \frac{\text{coherence}(a \to X) + \text{linkprob}(a)}{2}$$

Finally, pairs are pruned if score(a $\rightarrow$ X) < λ, where the threshold λ is set on a held-out set.

Neural Graph-based linking

More recent entity linking models are based on biencoders, encoding a candidate mention span, encoding an entity, and computing the dot product between the encodings. This allows embeddings for all the entities in the knowledge base to be precomputed and cached.



Let's sketch the ELQ linking algorithm, which is given a question q and a set of candidate entities from Wikipedia with associated Wikipedia text, and outputs tuples (e,ms,me) of entity id, mention start, and mention end

Neural Graph-based linking

Entity Mention Detection To get an h-dimensional embedding for each question token, the algorithm runs the question through BERT in the normal way:

$$[\mathbf{q}_1 \cdots \mathbf{q}_n] = \text{BERT}([\text{CLS}]q_1 \cdots q_n[\text{SEP}])$$

It then computes the likelihood of each span [i, j] in q being an entity mention, in a way similar to the span-based algorithm we saw for the reader above. First we compute the score for i/ j being the start/end of a mention:

$$s_{\text{start}}(i) = \mathbf{w}_{\text{start}} \cdot \mathbf{q}_i, \qquad s_{\text{end}}(j) = \mathbf{w}_{\text{end}} \cdot \mathbf{q}_j,$$

Neural Graph-based linking

where $\mathbf{w}_{\text{start}}$ and $\mathbf{w}_{\text{end}}$ are vectors learned during training. Next, another trainable embedding, $\mathbf{w}_{\text{mention}}$ is used to compute a score for each token being part of a mention:

$$s_{\text{mention}}(t) = \mathbf{w}_{\text{mention}} \cdot \mathbf{q}_t$$

Mention probabilities are then computed by combining these three scores:

$$p([i,j]) = \sigma\left(s_{\text{start}}(i) + s_{\text{end}}(j) + \sum_{t=i}^{j} s_{\text{mention}}(t)\right)$$

Neural Graph-based linking

To link mentions to entities, we next compute embeddings for each entity in the set E = e1,⋯ , ei ,⋯ , ew of all Wikipedia entities. For each entity ei we'll get text from the entity's Wikipedia page, the title t(ei) and the first 128 tokens of the Wikipedia page which we'll call the description d(ei). This is again run through BERT, taking the output of the CLS token BERT[CLS] as the entity representation:

$$\mathbf{x}_e = \text{BERT}_{[\text{CLS}]}([\text{CLS}]t(e_i)[\text{ENT}]d(e_i)[\text{SEP}])$$

Neural Graph-based Linking

Mention spans can be linked to entities by computing, for each entity e and span [i, j], the dot product similarity between the span encoding (the average of the token embeddings) and the entity encoding.

$$\mathbf{y}_{i,j} = \frac{1}{(j-i+1)} \sum_{t-i}^{j} \mathbf{q}_t$$

$$s(e,[i,j]) = \mathbf{x}_e \mathbf{y}_{i,j}$$

Finally, we take a softmax to get a distribution over entities for each span:

$$p(e|[i,j]) = \frac{\exp(s(e,[i,j]))}{\sum_{e' \in \mathcal{E}} \exp(s(e',[i,j]))}$$

The ELQ mention detection and entity linking algorithm is fully supervised

Knowledge Based QA

- While an enormous amount of information is encoded in the vast amount of text on the web, information 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 that answered questions from a structured database of baseball games and stats.

Knowledge Based QA

- There are two common paradigms are used for knowledge-based QA.
- The first, **graph-based QA**, models the knowledge base as a graph, often with entities as nodes and relations or propositions as edges between nodes.
- The second, QA by **semantic parsing**

Knowledge Based QA - Graph Based using RDF triple stores

- Let’s introduce the components of a simple knowledge-based QA system after entity linking has been performed.
- We’ll focus on the very simplest case of graph-based QA, in which the dataset is a set of factoids in the form of RDF triples, and the task is to answer questions about one of the missing arguments.
- An RDF triple is a 3-tuple, a predicate with two arguments, expressing some simple relation or proposition.

Knowledge Based QA - Graph Based using RDF triple stores

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?”.

Knowledge Based QA - Graph Based using RDF triple stores

A number of such question datasets exist.

SimpleQuestions contains 100K questions written by annotators based on triples from Freebase. For example, the question ”What American cartoonist is the creator of Andy Lippincott?”. was written based on the triple (andy Lippincott, character created by, garry trudeau).

FreebaseQA, aligns the trivia questions from TriviaQA and other sources with triples in Freebase, aligning, for example, the trivia question “Which 18th century author wrote Clarissa (or The Character History of a Young Lady), said to be the longest novel in the English language?” with the triple (Clarissa, book.written-work.author, Samuel Richardson).

Knowledge Based QA - Graph Based using RDF triple stores

Another such family of datasets starts from WEBQUESTIONS, which contains 5,810 questions asked by web users, each beginning with a wh-word, containing exactly one entity, and paired with handwritten answers drawn from the Freebase page of the question's entity.

WEBQUESTIONSSP augments WEBQUESTIONS with human-created semantic parses (SPARQL queries) for those questions answerable using Freebase.

COMPLEXWEBQUESTIONS augments the dataset with compositional and other kinds of complex questions, resulting in 34,689 questions, along with answers, web snippets, and SPARQL queries.

Knowledge Based QA - Graph Based using RDF triple stores

Let's assume we've already done the stage of entity linking introduced in the prior section. Thus we've mapped already from a textual mention like Ada Lovelace to the canonical entity ID in the knowledge base. For simple triple relation question answering, the next step is to determine which relation is being asked about, mapping from a string like "When was ... born" to canonical relations in the knowledge base like birth-year. We might sketch the combined task as:

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

"What is the capital of England?" → `capital-city(?x, England)`

Knowledge Based QA - Graph Based using RDF triple stores

The next step is relation detection and linking. For simple questions, where we assume the question has only a single relation, relation detection and linking can be done in a way resembling the neural entity linking models: computing similarity (generally by dot product) between the encoding of the question text and an encoding for each possible relation.

Knowledge Based QA - Graph Based using RDF triple stores

Ranking of answers : Most algorithms have a final stage which takes the top j entities and the top k relations returned by the entity and relation inference steps, searches the knowledge base for triples containing those entities and relations, and then ranks those triples.

This ranking can be heuristic, for example scoring each entity/relation pairs based on the string similarity between the mention span and the entities text aliases, or favoring entities that have a high in-degree (are linked to by many relations).

Or the ranking can be done by training a classifier to take the concatenated entity/relation encodings and predict a probability.

Knowledge Based QA - Semantic Parsing

The second kind of knowledge-based QA uses a semantic parser to map the question to a structured program to produce an answer. These logical forms can take the form of some version of predicate calculus, a query language like SQL or SPARQL, or some other executable program

The logical form of the question is thus either in the form of a query or can easily be converted into one (predicate calculus can be converted to SQL, for example). The database can be a full relational database, or some other structured knowledge store

Knowledge Based QA - Semantic Parsing

Semantic parsing algorithms can be supervised fully with questions paired with a hand-built logical form, or can be weakly supervised by questions paired with an answer (the denotation), in which the logical form is modeled only as a latent variable.

For the fully supervised case, we can get a set of questions paired with their correct logical form from datasets like the GEOQUERY dataset of questions about US geography (Zelle and Mooney, 1996), the DROP dataset of complex questions (on history and football games) that require reasoning, or the ATIS dataset of flight queries, all of which have versions with SQL or other logical forms.

Knowledge Based QA - Semantic Parsing

The task is then to take those pairs of training tuples and produce a system that maps from new questions to their logical forms. A common baseline algorithm is a simple sequence-to-sequence model, for example using BERT to represent question tokens, passing them to an encoder-decoder.

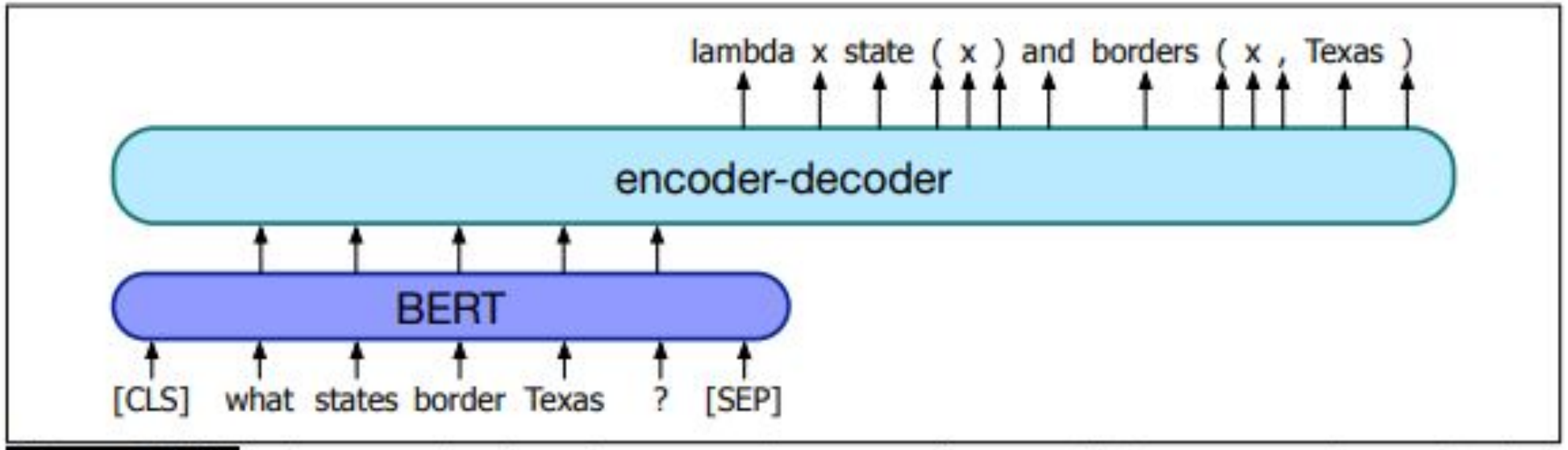


Figure 14.15 An encoder-decoder semantic parser for translating a question to logical form, with a BERT pre-encoder followed by an encoder-decoder (biLSTM or Transformer).

Knowledge Based QA - Semantic Parsing

PES UNIVERSITY

Question	Logical form
What states border Texas?	$\lambda x.\text{state}(x) \wedge \text{borders}(x, \text{texas})$
What is the largest state?	$\text{argmax}(\lambda x.\text{state}(x), \lambda x.\text{size}(x))$
I'd like to book a flight from San Diego to Toronto	SELECT DISTINCT f1.flight_id FROM flight f1, airport_service a1, city c1, airport_service a2, city c2 WHERE f1.from_airport=a1.airport_code AND a1.city_code=c1.city_code AND c1.city_name= 'san diego' AND f1.to_airport=a2.airport_code AND a2.city_code=c2.city_code AND c2.city_name= 'toronto'
How many people survived the sinking of the Titanic?	(count (!fb:event.disaster.survivors fb:en.sinking_of_the_titanic))
How many yards longer was Johnson's longest touchdown compared to his shortest touchdown of the first quarter?	ARITHMETIC diff(SELECT num(ARGMAX(SELECT)) SELECT num(ARGMIN(FILTER(SELECT))))

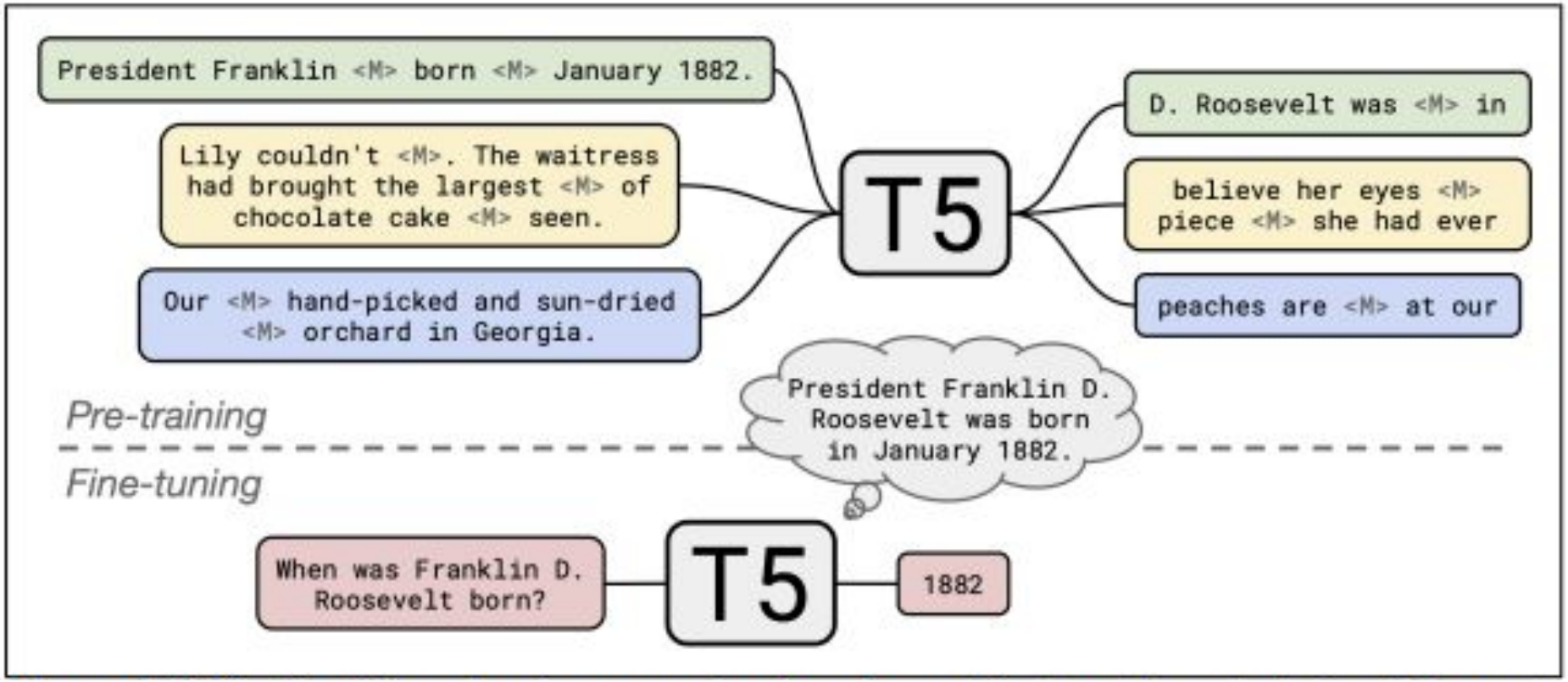
Sample logical forms produced by a semantic parser for question answering,

Pretrained Models for QA

- An alternative approach to doing QA is to query a pre-trained language model, forcing a model to answer a question solely from information stored in its parameters.

- For example the T5 language model, which is an encoder decoder architecture pretrained to fill in masked spans

Question Answering



The T5 system is an encoder-decoder architecture. In pretraining, it learns to fill in masked spans of task (marked by) by generating the missing spans (separated by) in the decoder. It is then fine-tuned on QA datasets, given the question, without adding any additional context or passages

Question Answering

- Roberts et al. (2020) then finetune the T5 system to the question answering task, by giving it a question, and training it to output the answer text in the decoder. Using the largest 11-billion-parameter T5 model does competitively, although not quite as well as systems designed specifically for question answering.

- Language modeling is not yet a complete solution for question answering; for example in addition to not working quite as well, they suffer from poor interpretability (unlike standard QA systems, for example, they currently can't give users more context by telling them what passage the answer came from). Nonetheless, the study of extracting answers from language models is an intriguing area for future question answer research.

Watson DeepQA system

PES UNIVERSITY

Watson DeepQA system from IBM won the Jeopardy! challenge in 2011

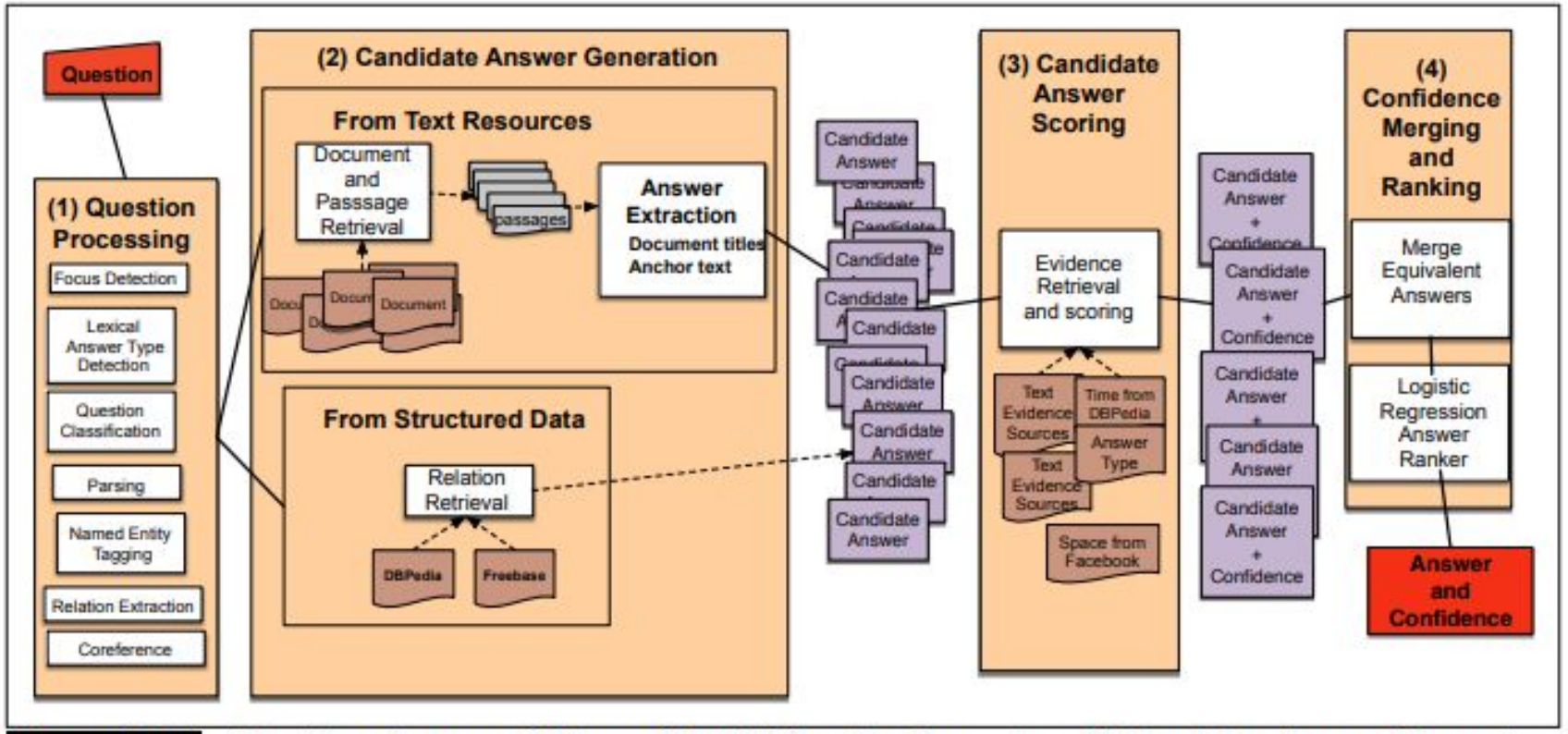


Figure 14.17 The 4 broad stages of Watson QA: (1) Question Processing, (2) Candidate Answer Generation, (3) Candidate Answer Scoring, and (4) Answer Merging and Confidence Scoring.

Watson DeepQA system

Let's consider how it handles these Jeopardy! examples, each 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.

Watson DeepQA System

PES UNIVERSITY

1. Question Processing

In this stage 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).

The question focus, shown in bold in both examples, is extracted. The focus is the string of words in the question that corefers with the answer. It is likely to be replaced by the answer in any answer string found and so can be used to align with a supporting passage. In DeepQA the focus is extracted by handwritten rules—made possible by the relatively stylized syntax of Jeopardy! questions—such as a rule extracting any noun phrase with the determiner “this” as in the Conan Doyle example, and rules extracting pronouns like she, he, hers, him, as in the poet example.

Watson DeepQA System

The lexical answer type (shown in blue above) is a word or words which tell lexical answer type us something about the semantic type of the answer. Because of the wide variety of questions in Jeopardy!, DeepQA chooses a wide variety of words to be answer types, rather than a small set of named entities. 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

Watson DeepQA System

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 logistic regression classifier that can return a probability as well as a lexical answer type. These answer types will be used in the later ‘candidate answer scoring’ phase as a source of evidence for each candidate.

Watson DeepQA System

Relations like the following are also extracted:

authorof(focus,“Songs of a sourdough”)

publish (e1, he, “Songs of a sourdough”)

in (e2, e1, 1907)

temporallink(publish(...), 1907)

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

Watson DeepQA System

PES UNIVERSITY

2. Candidate Answer Generation

Next we combine the processed question with external documents and other knowledge sources to suggest many candidate answers from both text documents and structured knowledge bases.

To extract answers from text DeepQA uses simple versions of Retrieve and Read. For example for the IR stage, DeepQA generates a query from the question by eliminating stop words, and then upweighting any terms which occur in any relation with the focus.

Watson DeepQA System

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 passed to a standard IR system: (2.0 Robert Redford) (2.0 Paul Newman) star depression era grifter (1.5 flick)

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.

Watson DeepQA System

Once we have a set of passages, we need to extract candidate answers. If the document happens to be 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 and used to point to a URL in an HTML page), or to extract all noun phrases in the passage that are Wikipedia document titles.

Watson DeepQA System

PES UNIVERSITY

3.Candidate Answer Scoring

Next DeepQA uses many sources of evidence to score each candidate.
This includes a classifier that scores whether the candidate answer can be interpreted as a subclass or instance of the potential 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 hyponymy, or synonymy link, in this case finding hyponymy between “Symptom” and “Condition”.

Watson DeepQA System

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

4. Answer Merging and Scoring

DeepQA finally merges equivalent candidate answers. 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 example using the anchor dictionaries described above for entity linking, which will list many synonyms for Wikipedia titles (e.g., JFK, John F. Kennedy, Senator John F. Kennedy, President Kennedy, Jack Kennedy). We then merge the evidence for each variant, combining the scoring feature vectors for the merged candidates into a single vector.

Watson DeepQA System

Now we have a set of candidates, each with a feature vector. A classifier takes each feature vector and assigns a 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 learns 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.

Watson DeepQA System

DeepQA's fundamental intuition is thus to propose a very large number of candidate answers from both text-based and knowledge-based sources and then use a rich variety of evidence features for scoring these candidates.