Ask Your TV: Real-Time Question Answering with Recurrent Neural Networks

Ferhan Ture
Comcast Labs
1110 Vermont Ave NW Washington, DC 20005
ferhan_ture@cable.comcast.com

Oliver Jojic
Comcast Labs
1110 Vermont Ave NW Washington, DC 20005
oliver_jojic@cable.comcast.com

Voice-based interfaces are very popular in today's world, and Comcast customers are no exception. Usage stats show that our new X1 TV platform receives millions of voice queries per day. As a result, expanding the coverage of our voice interface provides a critical competitive advantage, allowing customers to speak freely instead of having to stick to a rigid set of commands. The ultimate objective is to provide a more natural user experience and increase access to our knowledge graph (KG) and entertainment platform.

We describe a real-time factoid question answering (QA) system, using our internal KG for training (i.e., generating labeled example question-answer pairs) and for retrieval at test time. We hope that this will inspire other companies to take advantage of readily available unlabeled data, machine learning and search technologies to build products that can improve customer experiences. Our approach consists of two steps: First, two neural network models are trained to *predict* a structured query from the free-form input question. Then, a *search* through all facts in the KG retrieves answers consistent with the structured query.

Related work: Iyyer et al. [3] assume the answer is an entity, treating QA as classification, which cannot scale to millions of entities. Also, this does not cover cases where the answer is not a named entity. Dong et al. [2] search for answers within two hops of the target node in a KG, but they assume that the target entity is provided to them. Entity resolution is, in our experience, the most difficult part of factoid QA, especially when the question consists largely of a title and/or name. Memory Networks and variations have worked quite well on QA benchmark datasets, scaling up to similar-sized KGs [1]. However, the main focus is to rerank candidate facts with respect to a given question. Candidate facts are generated based on an n-gram match between question and named entities in the KG. Instead, we train a model to extract entity text from the question, which reduces latency and decouples entity detection from the KG. A KG in the entertainment domain is likely to be dominated by the relation type denoting that a person took part in a watchable (e.g., actress appeared in movie). Due to this, simply visiting all nodes within K hops of an entity might increase latency significantly. We train a separate model to predict the relation type(s).

From Question to Structured Query: Our internal database is converted into a *knowledge graph*, consisting of a large set of *facts* — a binary [subject, relation, object] or ternary [subject, relation, attribute, object] relationship. We focus on *first-order questions*, where the answer can be retrieved from a single fact (e.g., "How old is Tom Hanks?" can be answered by the fact [Tom Hanks,birthDate,7/9/1956]).

Our approach is based on converting the input question into a structured query, which can be ran against the knowledge graph to retrieve the fact(s) that answer(s) the question (e.g., the question above would be converted to [Tom Hanks, birthDate, ?]). This process can be modularized into two parts: *entity detection* and *relation prediction*. In the former, the objective is to *tag* whether each question word is part of an entity, and optionally further divide into *subject*, *object*, and *attribute* classes. In the latter, the objective is to *classify* the question into one of the K relation types.

We used recurrent neural networks (RNN) to tackle both tasks, hence dependencies across words in the question are naturally modeled without the need for feature engineering. Certain variations of RNNs, such as long short-term memory (LSTM), can theoretically learn contextual features that are useful for the task, while ignoring irrelevant parts. RNNs' effectiveness in exploiting wide textual context for classification or tagging has been shown in numerous cases.

Each question word passes through an embedding lookup layer before the recurrent layer(s). We fixed this layer based on the pre-trained 300-dimensional Google News embedding,[1] which is followed by one or more recurrent layers. In tagging, for each question word, output of the last RNN layer is projected to one of the 4 tags (subject, object, attribute, none) via a fully connected layer. In classification, words are processed through recurrent units sequentially, and the question is classified at the end, with the hidden layer from the entire question as part of the input. In both cases, parameters are learned via stochastic gradient descent, using categorical cross-entropy as objective. In order to handle variable-length input, we limit to 15 tokens and prepend a special pad word if input has fewer.

A large training set is key to success with RNNs — this is especially expensive for QA, since domain-relevant questions need to be generated and annotated with tags at the word-level, as well as a relation type at the question-level. In

[1] word2vec.googlecode.com Out-of-vocabulary (OOV) words are randomly sampled from a uniform distribution.

SIGIR '16 July 17-21, 2016, Pisa, Italy

ACM ISBN 978-1-4503-4069-4/16/07.

DOI: http://dx.doi.org/10.1145/2911451.2926729

practice, such data is rarely available at development time. As a solution to this cold start problem, we generated training data directly from the KG in a synthetic manner, and augmented further to improve robustness. After identifying the set of relation types in the KG, we manually created question templates for each. For example, for the `birthDate` relation, one such template is "How old is *subj*?". Using these templates, we can generate questions for each fact in the KG, by instantiating the variables (e.g., *subj*="Tom Hanks"). This process requires significantly less labor than annotating millions of training examples.[2]

Synthetic data is easy to create, but the lack of lexical diversity might cause overfitting, especially with complex models. For example, an RNN will learn that questions *always* follow a certain template, and therefore fail even with a slight variation at test time. This aspect of QA is often ignored in academic publications, and we are not aware of benchmark datasets that focus on noisy factoid questions. For our product, this is especially critical, since many utterances are not well-formed. In order to tackle this, we expand certain domain-specific words with their synonyms, simply by using a dictionary. Additionally, we add noise by randomly (a) switching the plurality of nouns, (b) changing the tense of verbs, and (c) dropping context words. We experiment with different ratios of synonym and noise expansion.

Searching the Knowledge Graph: In order to make our KG searchable (given a structured query), we build multiple indexes: an inverted index for each entity type I_{subj}, I_{obj}, I_{attr} scores all n-grams of an entity using BM25 (e.g., "lost world" is a 2-gram for the entity "Lost World: Jurassic Park", so the index creates a mapping from the former to the latter with an associated score). A separate index I_{reach} maps each entity node s to all nodes e' that are reachable (a single hop away). Search for QA works as follows: Given a new question, we run the two RNN models to construct the structured query q (e.g., {*rel*=director, *subj*=lost world}). We go through all entity n-grams t' (starting from the entire text "lost world"), and find all matching entity nodes through $I_{\text{subj}}(t')$ (e.g., ["The Lost World", "Lost World: Jurassic Park", ...]). For each matching node q_e, we find all nodes reachable from it, $I_{\text{reach}}(q_e)$, and filter out ones where the path is not consistent with q Remaining nodes are candidate answers, scored by $BM25(t, q_e)$.

Evaluation: Our KG was constructed incrementally, first initialized with all direct facts about the most popular 64K entities. This was expanded twice, to include all indirect facts (i.e., direct facts about neighbor entities). As a result, we have 18.6M facts about 3.9M entities, from which 13.9M synthetic training examples were generated, as well as 4.5M/9.8M more through synonym/noise addition.

The KG is indexed into two files, each around 2GB, which is easily stored in memory. This allows very high latency — below 100ms per question on a research server running an Intel Xeon 2.66GHz processor. Each model takes around 40ms to predict, and the remaining time is used for search.

A challenge related to the entertainment domain is the large number of rarely used and foreign named entity words. Even with the 3 million pre-trained word embeddings, our training set has a 48% OOV rate. How much this impacts the end task, and whether there is a good solution to train embeddings for these words, are still open questions.

[2]There are 55 relation types in our entertainment KG.

We experimented with a variety of parameters and tested both tasks on held-out data, using the open-source deep learning toolkit Keras (http://keras.io). Due to space limitations, we will mention few key findings. With only synthetic questions, both tasks only make mistakes for the few ambiguous questions (e.g., author vs screenwriter vs writer). A more realistic case is when we add noise — in this case, classification accuracy drops to 96.2% (single LSTM layer with 20% drop out) while tagging accuracy remains to be almost 100% F1-score (two LSTM layers with 20% drop out). To simulate the worst-case scenario in which the test set is not just a noisy variation of an observed question type, but a completely new form of question, we separated the synthetic question templates into train, validation, and test sets. Classification accuracy decreased all the way down to 52.2%, using a single bidirectional LSTM with 10% drop out. The tagging model must have better generalization properties, since it was able to obtain an F1 of 91% for the *attribute* tag, and 88% for *subject* tags. The 45% F-score on *object* was due to a single type of unseen question — while this is a good exercise to understand the weakness of our approach, continuing the training process with new synthetic examples of the incorrect question type will easily fix the problem.

Conclusions and Future work: We described a simple yet effective approach for factoid question answering: a low-latency, scalable, and generalizable approach using minimal labor (no time spent on feature engineering!), based on a publicly available free machine learning toolkit and relatively cheap hardware. Our system is in the development phase, as it is being tested for quality assurance. We are hoping to bring it into production in the near future, and provide an innovative way for our customers to engage with their TV. We anticipate that our solution will add business value to our company, while building motivation for future research in the IR/NLP/ML space.

Next step is to handle questions where there are multiple entities, connected by various relations (e.g., "Which action movie did Tom Cruise and Meg Ryan both appear in?"). We would need to modify the structured query and expand search to more than one hop. Another important extension is to train a separate model to learn-to-rank the answers output by the current approach. This could especially help with personalization. We are also eager to find out how we can take advantage of deeper networks for the purpose of QA. Ensemble methods can provide improvements, since many times two variations of the model make complementary predictions. On the other hand, deeper models can implicitly learn the equivalent of an ensemble model. A key feature of our approach is that the knowledge search is separated from structured prediction. However, incorporating what we know into the model can provide higher accuracy.

1. REFERENCES

[1] A. Bordes, N. Usunier, S. Chopra, and J. Weston. Large-scale simple question answering with memory networks. *CoRR*, abs/1506.02075, 2015.

[2] L. Dong, F. Wei, M. Zhou, and K. Xu. Question answering over freebase with multi-column convolutional neural networks. In *ACL*, 2015.

[3] M. Iyyer, J. L. Boyd-Graber, L. M. B. Claudino, R. Socher, and H. Daumé. A neural network for factoid question answering over paragraphs. In *EMNLP*, 2014.