Predicting answer types for question-answering

Ivan Bogatyy*
Google Inc.
1600 Amphitheatre pkwy
Mountain View, CA 94043
bogatyy@google.com

Abstract

We tackle the problem of answer type prediction, applying a transition-based neural network parser to understand the syntactic structure of a query and infer the focus word or phrase that contains the correct type. To achive this, we first come up with a framework to combine a dataset of labeled question-answer pairs and a type inventory into a dataset to learn types. Second, we fit a sigmoid prediction layer on top of query words and the inferred syntactic structure of a query to predict the correct answer types. Our model outperforms a logistic regression baseline that does not take into account the syntactic structure of the query.

1 Introduction and Related Work

Many of the question-answering systems capable of answering a broad range of world knowledge on a production scale tend to have modular architecture and rely heavily on information retrieval techniques, e.g. Watson [4] or Google's web-based answering engine (for a couple examples of the system at work, consider queries [who was the president during the moon landing] or [is eminem left handed]). These systems are known to benefit from answer type modeling [5]: limiting the possible set of candidates significantly increases both speed and quality. The $entity \leftrightarrow type$ models are extensively studied, using hypernyms [1], manually curated relationships [9] and Hearst pattens. The $query \leftrightarrow type$ relationships are much harder to infer and present an interesting research opportunity.

The goal of the project is to build a neural network model that, given a question, predicts its answer's type. For example, given a query [the tennis player who wore denim shorts], the model does not have to come up with the exact answer (Andre Agassi), but it has to predict some reasonable set of types ("human", "tennis player", "athlete", etc) that the correct answer has to belong to.

Overall, the approach is as follows. First, we implement a type matcher, which canonicalizes words and bigrams within the query and annotates them accordingly if they correspond to some types within the type inventory considered. For example, consider the query [who starred in that horror movie about lambs]. Here, "who" would be annotated with $/collections/people$, "movie" would be annotated with $/collections/movies$, and "horror movie" would be annotated with $/collections/horror\_movies$. We call such annotated spans (words or phrases) slots. Second, we create a labeled dataset of query - type pairs. Third, we use a globally normalized transition-based neural network parser [12] to use the syntactic structure of the query to enhance type prediction quality.

Regarding the third part, model selection, the core choice has been between an architecture based on a recurrent neural network, like [2] or [3], and a feed-forward transition-based architecture introduced in [10] and further developed in [11, 12]. Given that the current SOTA belongs to the globally normalized feed-forward transition-based parser [12], their model is hundreds of times faster than a recurrent architecture and is easily accessible with TensorFlow, it was our final choice.

*Work done as the final project for CS224D

2 Approach

2.1 Framework for mining correct types

Assume we have a type inventory (for example, Freebase / KnowledgeGraph collections [9]), a type matcher (which maps query tokens onto types within that inventory) and a labeled dataset of questions and their correct answers. We define the correct answer types for these questions to be as follows. First, we run a type matcher on the query, producing type annotations, e.g. [movies about presidents] - $/collections/movies$, $/collections/presidents$. Second, for every correct answer we find the types it belongs to. For example, "All the President's Men" - $/collections/movies$, $/collections/political\_thrillers$, "Abraham Lincoln: Vampire Hunter" - $/collections/movies$, $/collections/vampire\_movies$. Third, we define the correct types to be the intersection of these sets (that is, all the types found within the query that every correct answer belongs to).

2.2 Baselines

The first baseline we evaluated works as follows. We take all the type annotations produced by type matcher running on the query. The resulting zeros-and-ones vector is rescaled using only two universal real-valued parameters ($scale$ and $offset$), and is returned as the unscaled prediction. This is the equivalent of simply returning every type that the type matcher had found, with the slight modification of rescaling the output to avoid infinite loss for incorrectly predicting hard 0 or 1 probabilities. The two parameters of the model are learned, in all experiments settling around the values of 10.0 and 1.2, which translates into predicting the probability of $6.1 \cdot 10^{-6}$ for types absent in the query annotation and $1.2 \cdot 10^{-1}$ for types present in the query annotation.

The second baseline takes the same zeros-and-ones type vector from the type matcher as input, and uses logistic regression to predict every output type independently (an equivalent explanation would be in terms of a feed-forward neural network of depth 1). To make the model converge faster, it is initialized to the values replicating the first baseline.

2.3 Model

The core type prediction model investigated here uses the same data as the previous two baselines (generated by type matcher), and also features based on the syntactic structure of the query. Binary features are generated for the slots corresponding to the $ROOT$ and $NSUBJ$ of the sentence, and an integer feature is generated for every slot based on its depth in the syntactic tree. As mentioned before, we use globally normalized transition-based neural network parser [12, 13] to obtain the syntactic structure of the query.

To make the model converge faster and reduce the chances of being stuck in suboptimal local minima, the predictive layer is pre-initialized to replicate baseline 2 (basically making the prediction based only on type matcher output), and the syntactic parser is pre-initialized to use the parameters of Parsey McParseface [13], which is an English dependency parser trained on Treebank Union.

3 Experiments

3.1 Datasets

For the query-answer pairs, we have used SimpleQuestions [7]. For type inventories, we have experimented with curated Freebase/KnowledgeGraph collections database (abbreviated as KG below) and another internal proprietary type inventory based on Hearst patterns (abbreviated as HP). For both type inventories, the correct type labels are produced using the approach described above.

Let us consider some interesting statistics about the dataset obtained. The training part of SimpleQuestions contains roughly $76k$ queries and $37k$ unique answers. Of those, roughly $35k$ have KG types (the number is rather high because all the answers are from Freebase [9]), and roughly $28k$ have HP types. Of the $76k$ queries, only 1822 have no type-related annotations.

After we intersect the query annotations and correct answer types to produce the final dataset, of importance is exploring the distribution of the number of correct types per query. Queries with zero

associated correct types represent the portion of the dataset where answer type prediction cannot help question-answering in principle, at least with the setup explored here. For the other queries, the expectation is that most of the queries have only a few correct answer types, and in general the distribution follows a power law. The numbers indeed conform to that expectation. Figure 1 shows a histogram of the distribution (in log-scale, to make the right side of the plot legible). The raw numbers are as follows. For KG, $\{0 : 21570, 1 : 23870, 2 : 12996, 3 : 6248, 4 : 2819, 5 : 1930, 6 : 3058, 7 : 1651, 8 : 1028, 9 : 339, 10 : 202, 11 : 96, 12 : 27, 13 : 21, 14 : 13, 15 : 15, 16 : 6, 17 : 7, 18 : 4, 19 : 3, 20 : 1, 21 : 1, 22 : 1, 24 : 1, 25 : 1, 29 : 2\}$. The distribution is rather heavy-tailed, because the parser produced an extensive list of similar collections (e.g. song_writers, musical_artists, music_group_members, soul_artists, blues_artists, etc). For HP, the distribution is as follows: $\{0 : 33359, 1 : 29375, 2 : 10283, 3 : 2408, 4 : 394, 5 : 76, 6 : 14, 7 : 1\}$. This may appear surprising given HP type inventory is much larger, but is explained by the fact that the actual bottleneck is the type matcher (which, on the query side, produces significantly more KG annotations than HP annotations).

Uniting all KG and HP annotations of all queries, we obtain an type vocabulary of 15572 elements, to which we refer as "full type vocabulary". The number of types occurring among correct types is significantly smaller: 2526 for KG and 3425 for HP. After running into difficulties fitting models based on full type vocabulary into RAM, we restrict the model vocabularies to only the types that appear as correct at least once (referred as short vocabulary). While this restriction may potentially limit the expressive power of the models, we actually observed a slight quality gain for baseline models after this change.

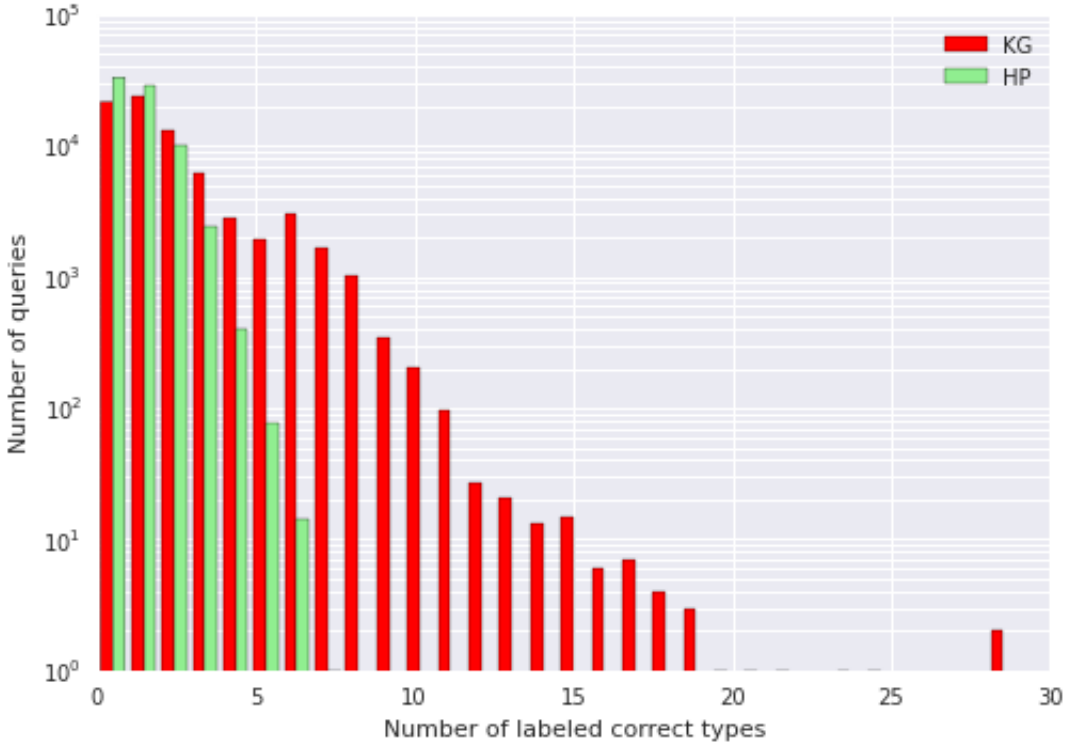

Figure 1: Number of queries per number of labeled correct types

3.2 Evaluation

The evaluation setup is typical for a non-exclusive multi-class prediction problem (since there may be more than one correct type, and recognizing a certain type as correct does not preclude other types from being correct and useful too). All the models we evaluated work by calculating unscaled

Table 1: Cross-enropy loss on KG-based dataset

MODEL	Full vocab	Short vocab	2+ slots	3+ slots
Rescaled type matcher	8.1	7.6	10.7	13.9
Logistic regression	4.0	3.8	5.0	6.1
Syntactic parser	Did not fit in RAM	**3.2**	**4.5**	**5.8**

Table 2: Cross-enropy loss on HP-based dataset

MODEL	Short vocab	2+ slots	3+ slots
Rescaled type matcher	2.0	2.3	2.5
Logistic regression	1.2	1.43	1.58
Syntactic parser	**0.7**	**0.9**	**1.1**

probabilities for every possible type. These probabilities are then normalized using the sigmoid function, and cross-entropy loss is evaluated.

3.3 Results

Tables 1 and 2 contain the results for KG-based evaluation and HP-based evaluation respectively. To provide a more nuanced look into the performance of the model, we also ran the training and evaluation process separately on certain slices of the data. More specifically, we selected only queries that have at least 2 different type-annotated spans (slots), and also the queries which have at least 3 slots. Such queries are the most interesting to us, as multiple slots present more interesting choices to the model.

We clearly see that for both type inventories and across all slices, the syntactic parser significantly outperforms the baseline. One peculiarity worth investigating is that for KG types, counter-intuitively, having more slots actually reduces the difference in performance between the baseline and the syntactic parser.

4 Conclusions and Future Work

We come up with a framework to infer $query \rightarrow type$ models, using only $query \rightarrow answers$ data, and explore the resulting distribution of correct types per query. The strength of this approach is that it generalizes significantly better than attempting to predict the answer itself.

We apply a transition-based neural network dependency parser, and show that models utilizing syntactic structure data significantly outperform models based only on $query\ word \leftrightarrow type$ mappings.

For future work, it seems important to investigate the phenomenon observed on KG slices: why the difference in performance between the baseline and the syntactic parser is reduced on queries with more slots (the expectation is that with more slots to choose from, the difference should be larger). It would be interesting to replicate the results on a dataset containing more complex queries (SimpleQuestions only contains answers from Freebase, which is a fairly strict limitation). It would also be interesting to compare the transition-based parser with a recurrent architecture.

References

[1] George A. Miller *WordNet: A Lexical Database for English.* Communications of the ACM Vol. 38, No. 11: 39-41, 1995.

[2] Wang Ling, Chris Dyer, Alan W Black, Isabel Trancoso, Ramon Fermandez, Silvio Amir, Luis Marujo, and Tiago Luis *Finding function in form: Compositional character models for open vocabulary word representation* In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pages 15201530, 2015.

[3] Oriol Vinyals, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton *Grammar as a foreign language.* In Advances in Neural Information Processing, Systems 28, pages 27552763, 2015.

[4] D. Ferrucci, E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A.A. Kalyanpur, A. Lally, J.W. Murdock, E. Nyberg, J. Prager, others *Building Watson: An overview of the DeepQA project*, AI Magazine 31(3), 59–79, American Association for Artificial Intelligence, 2010.

[5] JW Murdock, A Kalyanpur, C Welty, J Fan, DA Ferrucci, DC Gondek, L Zhang, H Kanayama *Typing candidate answers using type coercion*, IBM Journal of Research and Development 56(3/4), 7:1 - 7:13, IBM, 2012.

[6] Jonathan Berant, Andrew Chou, Roy Frostig, Percy Liang *Semantic Parsing on Freebase from Question-Answer Pairs*, EMNLP, 2013.

[7] Antoine Bordes, Nicolas Usunier, Sumit Chopra, Jason Weston *Large-scale Simple Question Answering with Memory Networks*, http://arxiv.org/pdf/1506.02075.pdf

[8] Hill, F. Cho, KH., Korhonen, A., and Bengio, Y. *Learning to Understand Phrases by Embedding the Dictionary*, Transactions of the Association for Computational Linguistics (TACL), 2016.

[9] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. *Freebase: a collaboratively created graph database for structuring human knowledge.* In SIGMOD 08, pages 12471250, New York, NY. ACM

[10] Danqi Chen, Christopher Manning. *A Fast and Accurate Dependency Parser using Neural Networks*, In EMNLP 2014

[11] David Weiss, Christopher Alberti, Michael Collins, and Slav Petrov. *Structured training for neural network transition-based parsing.* In Proc. ACL., 2014

[12] Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins *Globally normalized transition-based neural networks.* Available at arXiv.org as arXiv:1603.06042, March 2016.

[13] http://googleresearch.blogspot.com/2016/05/announcing-syntaxnet-worlds-most.html