Single-Relation Question Answering with Attention Mechanisms and Deep Neural Networks

David Golub
University of Washington
golubd@cs.washington.edu

Xiaodong He
Microsoft Research
xiaohe@microsoft.com

Abstract

We propose a novel model for open question answering that borrows ideas from neural machine translation to ”translate” a question into a predicate and entity. Using a combination of multilayer recurrent neural networks, convolutional neural networks, and attention mechanisms our model learns to align questions to entities and predicates in a knowledge base. We evaluate two versions of our model–one with character level embeddings and one with word hashing on the Facebook SimpleQuestion dataset. From a random sample of 200 predicates and entities in the dataset and a correct predicate/entity, we find that the character level model learns to correctly predict entities with 77% accuracy and entities with 94.3% accuracy, with a joint prediction accuracy of 74% (TODO: model with word hashing is still training). Finally, we describe a novel entity linking system that uses deep semantic similarity models that our system can be used for an end-to-end system to single-relation question answering (TODO: need to make entity linking system).

1 Introduction

Open-domain question answering remains a challenging problem of crucial importance, and even the simpler task of single-relation question answering is far from being solved. We define single-relation question answering to be questions that can be answered by querying a knowledge base with a single entity and predicate argument, such as ”When was the personal computer invented?”.

The primary challenge in single-relation question answering is how to align the question to its corresponding entity and predicate counterparts in a knowledge base. The many possible paraphrases of the same question make it difficult to construct this many-to-one mapping.

2 Previous Work and Our Approach

2.1 Previous Work

We extend the work of [1], where the authors tackle single-relation QA by first decomposing a question into two parts, followed by a sophisticated semantic similarity model to address the paraphrase challenge and align these parts to entity/predicates in a knowledge base. The previous work creates a hard decomposition of the question into a parse tree with two leaf nodes where one represents the entity mention and the other represents the relation pattern. Then it projects both candidate entities/predicates from a knowledge base and these question decompositions using a multilayer convolutional neural network into a latent semantic space and measures the semantic similarity of their latent semantic representations. During test time, one scores relational triples in the KB using these measures and select the top scoring relational triple to answer the question.

Since building the tree-based semantic parsing framework to have high precision can be a very challenging problem, the previous work enumerates over all possible parse trees–or ”hard” alignments and relies on the power of its semantic similarity models for the paraphrase subchallenge. However, enumerating over all these trees is computationally expensive as one has to feed-forward through these convolutional neural nets for each of the trees and may also reduce the overall precision of the final model as these noisy examples produce additional opportunities for incorrect alignment.

2.2 Our Approach and connection to NMT

In order to address the issue, we borrow from the work of [2] on machine translation where the authors introduce an extension to the encoder–decoder model which learns to align and translate jointly. Each time the proposed model generates a word in a translation, it "soft-searches" for a set of positions in a source sentence which are most relevant.

We use this idea of a "soft" alignment where an attention-based LSTM encoder/decoder network selectively focuses on portions of the question that it believes best correspond to the predicates, and then those portions that best correspond to the entities. Rather than explicitly assigning each word in the question as an entity mention or a relation pattern in a parse and enumerating all of these parses, our approach assigns "soft" alignment scores and allows us to predict the most likely entity/predicates in only two passes over the question.

3 Model

We now describe the architecture of both the character and word-hashing models. The character level model is inspired by recent work by [3] where the authors tackled a variety of tasks ranging from classification to sentiment analysis from the character level and had convincing results. We assume that an entity-linking system already provides the model with predicates and entities.

Consequently, for both of the character and word-hashing models our inputs are a question q, a list of candidate predicates $p_{1..n}$ and a list of candidate entities $e_{1..m}$.

3.1 Embeddings and preprocessing

As the character and word-hashing models only differ in how they preprocess the inputs and produce the embeddings for the questions, entities, and predicates, we describe how they do it.

The character level model does no tokenization on the input, and has a separate embedding for each character. Namely, if we let the embedding matrix be E and the characters in a string be $y_{1..n}$ the output embeddings for the i^{th} character are $E \times y_i$, where y_i is a one-hot encoding for the respective character.

To generate the embeddings, we first tokenize the input, and then follow the word-hashing model follows the procedure in [4] where each word is represented by a count-vector of its respective trigrams, followed by a projection matrix $R^{V \times h'}$ where V is the number of unique character trigrams and h' is the embedding size for each word. To tokenize the input, we replace all special and non-alphanumeric characters with a white space and convert all characters to lowercase. Then, rather than having an embedding for each character, we have word-level embeddings to represent each entity, predicate or question.

For both models, we use separate embedding matrices for the questions, predicates, and entities with no shared parameters.

3.2 General architecture

We feed the embeddings of the question into a two-layered gated-feedback LSTM as described in [5] and take the outputs at each time step as the final embeddings for the question. These embeddings are the contexts for our attention LSTM.

To produce the latent semantic representations for the entity/predicates, we feed the character-level embeddings through a temporal convolutional neural network as described below:

$$f(x_{1...n}) = tanh(W_3 * max(tanh(W_2 * conv(\tanh W_1 * conv(x_{1..n}))))) \tag{1}$$

Where n is the number of characters, $f(x_{1...n})$ has size out, W_3 has size R^{oxh}, $conv$ represents a temporal convolutional neural network, and max represents a max pooling layer in the temporal direction, and x_i represents the embedding of the i^{th} character of the entity/predicates.

Let $y_{1...n}$ be the latent semantic representations of the candidate predicates where n is the number of candidate predicates, y_i = $f(y_i^{1...|z_i|})$. Similarly, let $z_{1..m}$ be the latent semantic representation of the candidate entities where m is the number of candidate entities.

The attention lstm decoder can be thought of as a function:

$$h_i, c_i = f(h_{i-1}, c_{i-1}, x_i, ctx_{i-1}). \tag{2}$$

where h_i, c_i are the RNN hidden/cell states for time i, x_i is the input at time i, and cxt_i are the context vectors at time i. For the attention lstm we follow the attention mechanism as described in [2] with the modification that $v_a \in R^{n \times n'}$ so consequently $\alpha_{ij} \in R^n$, rather than R.

Namely, h_i, c_i are computed as follows

$$i_t = \sigma(W_{ix}x_t + W_{im}m_{t-1} + W_{ic}c_{t-1} + W_{ictx}ctx_t + b_i) \tag{3}$$

$$f_t = \sigma(W_{fx}x_t + W_{mf}m_{t-1} + W_{cf}c_{t-1} + W_{ctxf}ctx_t + b_f) \tag{4}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g(W_{cx}x_t + W_{cm}m_{t-1} + W_{ctxm}ctx_t + b_c) \tag{5}$$

$$o_t = \sigma(W_{ox}x_t + W_{om}m_{t-1} + W_{oc}c_t + +W_{ctxm}ctx_t + b_o) \tag{6}$$

$$h_t = o_t \odot h(c_t) \tag{7}$$

$$\tag{8}$$

This is the same as an LSTM with no context vectors, except there is an additional term ctx which represents the encoding of the context vectors for the question.

The context vector ctx_t are recomputed at each step by the alignment model:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} \circ h_j, \tag{9}$$

where

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})} \tag{10}$$

$$e_{ij} = v_a^\top \tanh(W_a s_{i-1} + U_a h_j), \tag{11}$$

and $v_a \in R^{n \times n'}$, $W_a \in R^{n' \times n}$ and $U_a \in R^{n' \times n}$ are weight matrices.

The matrices (e.g. W_{ix} is the matrix of weights from the input gate to the input), the b terms denote bias vectors (b_i is the input gate bias vector), σ is the logistic sigmoid function, and i, f, o and c are respectively the input gate, forget gate, output gate and cell activation vectors, all of which are the same size as the cell output activation vector m, $\odot$ is the element-wise product of the vectors and g and h are the cell input and cell output activation functions, generally $tanh$.

Letting the outputs of the lstm at times $1, 2$ be o_1 and o_2 respectively, we want to maximize the following values:

$$\frac{exp(\lambda * cos(o_1, y_{corr}))}{\sum_{i=1}^{n} exp(\lambda * cos(o_1, y_i))} * \frac{exp(\lambda * cos(o_2, z_{corr}))}{\sum_{i=1}^{n} exp(\lambda * cos(o_2, z_i))}. \tag{12}$$

where λ is a constant and $corr$ is the index of the correct predicate/entity.

For our loss function we use a cross-entropy criterion.

At the time $t = 1$ we use a special input $x^* = \vec{0}$, whereas at time $t = 2$ we use the input y_{corr} during training and y_{pred} during testing where $pred = \arg\max_{i \in 1...n} exp(\lambda * cos(o_1, y_i))$.

4 Dataset and Results

We evaluate our models on the SimpleQuestions dataset [6] which consists of 108,442 single-relation questions and their corresponding source entity, predicate, target entity triples from Freebase.

4.1 Preprocessing and setup

For the predicate name we use the relative path as the name as directly seen in freebase, i.e `www.freebase.com/film/film_genre/films_in_this_genre` turns into `film/film_genre/films_in_this_genre`. For the entity name we query the freebase API and take the name as seen from Freebase. We currently do not distinguish between different entities that have the same name, we leave that work for a candidate generation/entity linking system and future papers.

To test our model, we randomly sample 200 entities and predicates from the test set in addition to the correct entity/predicate for each question. Consequently, we use our model to compute the most likely entity/predicate from these candidates, and measure the predicate, entity and joint prediction accuracy. The dataset consists of 21,687 test, 10,845 validation and 75,910 train questions respectively.

4.2 Different types of attention

We experiment with three different types of attention. We look at fine grained attention where each dimension of each hidden vector is multiplied by a scalar, coarse grained attention where the entire hidden vector is multiplied by a single scalar, and fine0grained attention with dropout. The hidden vectors in an LSTM is often treated as a cohesive unit (TODO provide citation) and each dimension individually is not assumed to be easily interpretable without considering its neighbors, so we explore different levels of attention to determine the validity of this assumption.

4.3 Results

We evaluate both our word-hashing and character level model. We train the character level model for 7 days on a single Tesla gpu (don't remember how long trained the word hashing level model). We find that after 11 epochs our word hashing model (which still hasn't converged) has 52.34% joint accuracy with 78.68% accuracy on predicting the predicate and 63.32% accuracy on predicting the entity. We found that our character-level model had a joint accuracy of 74.5%, with 76.8% accuracy on predicate prediction after 11 epochs and 95.67% accuracy on entity prediction.

4.3.1 Performance differences between character and word-hashing model

We found that our character-level model was able to predict the entity name with a significantly higher accuracy but struggled with predicting the predicate and peaked at 77% accuracy. In terms of convergence time, we found that the character level model was able to converge to a much higher entity prediction accuracy after fewer epochs, whereas the word-hashing model was able to converge to a much higher level of predicate accuracy much faster, but struggled with predicting the entity. One possible explanation is the large number of different, disjoint embeddings that the word-hashing model needs to learn to predict entities and may need more training examples to cover all of these possibilities. We display all of these results in 1 and 2.

4.3.2 Model compactness

We note that, keeping all other parameters fixed and only changing between character embeddings and word-level hashing, our character-level model 1,201,008 parameters as opposed to the 7,014,008 of our word-hashing model, an 83% reduction in number of parameters.

4.4 Qualitative results

It appears to be a very challenging task for a model to learn higher-order representations of As the SimpleQuestions dataset contains a large number of entities whose names appear directly in the question, we wanted to see whether the character level model learns to distinguish entity words from other words in a sentence or simply learns a form of string-matching between words in a sentence and entity names.

To test this out, we randomly sampled questions from the training set and as candidate entities, chose random n-grams from the sentence as well as the "true" entity.

Table 1: Results for character level model

Epoch	Joint Accuracy	Predicate Accuracy	Entity Accuracy
2	0.4327	0.5481	0.6942
5	0.6485	0.6916	0.9060
8	0.7152	0.7433	0.9434
11	0.7456	0.7681	0.9567

Table 2: Results for word hashing model

Epoch	Joint Accuracy	Predicate Accuracy	Entity Accuracy
2	0.0869	0.7042	0.1038
5	0.2260	0.7394	0.3267
8	0.3855	0.7720	0.4681
11	0.5234	0.7868	0.6332

4.4.1 Example sentences and candidates that model was able to distinguish

TODO

4.4.2 Attention visualizations for various questions

TODO

5 Future Results and Entity Linking

5.1 Entity Linking for SimpleQuestions

To provide a more fair comparison to the model developed in [6], we developed an entity-linking system to rerank candidate entities based off of their names and dominant freebase types.

5.1.1 Candidate Generation

We split up each question into 1..n-grams and queried the Freebase API for all possible entities whose name could be represented with these n-grams. This means, as opposed to Freebase 2M as used in [6], (TODO: find it to see if it exists), we queried entire freebase. We then extracted all predicates for all of these possibile entities, which gave us candidate triplets (entity1, predicate1, entity2) where ”entity1” is the source entity and ”entity2” is the target entity that can be reached from ”entity1” through predicate ”predicate1”.

5.1.2 Candidate Pruning

We used the same DSSM architecture as we did to embed the predicates/entities to embed both the question and names of the candidate entities in a latent semantic space. We also experimented with LSTM encoder architectures for the candidate entities, but found that our implementation took over 14 hours per an epoch (as opposed to 2 hours for the convolutional DSSM architecture) and had considerably more memory usage.

We used a bag of words representation to represent the type of the entity, i.e. ”`www.freebase.com/film/film_genre/films_in_this_genre`, `www.freebase.com/common/common_type/object`)_in_this_genre” would be represented by the sum of one-HOT word vectors. We then concatenated the type of entity to the latent semantic representation of the name and optimized the cosine similarity between this concatenated vector to the question-vector. I.e.

$$\frac{exp(\lambda * cos(o_1, y_{corr}))}{\sum_{i=1}^{n} exp(\lambda * cos(o_1, y_i))} \tag{13}$$

. For our training procedure we used negative sampling and a cross-entropy objective.

5.2 Results

To get the final results, we fed these candidates to our question answering models. TODO: get results! (Character-level model is still training).

Acknowledgments

Not sure what to put here.

References

[1] Wen tau Yih, Xiaodong He, and Christopher Meek. Semantic parsing for single-relation question answering. In *Proceedings of ACL*. Association for Computational Linguistics, June 2014.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014.

[3] Xiang Zhang and Yann LeCun. Text understanding from scratch. *CoRR*, abs/1502.01710, 2015.

[4] *Learning Deep Structured Semantic Models for Web Search using Clickthrough Data*. ACM International Conference on Information and Knowledge Management (CIKM), October 2013.

[5] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Gated feedback recurrent neural networks. *CoRR*, abs/1502.02367, 2015.

[6] Antoine Bordes, Nicolas Usunier, Sumit Chopra, and Jason Weston. Large-scale simple question answering with memory networks. *CoRR*, abs/1506.02075, 2015.